Лабораторна робота №4

Тема: Створення класів в C#. Поліморфізм.

Мета: Вивчити особливості роботи з власними класами у C#.

Завдання: Здобути навички створення, відлагодження та тестування проектів обробки даних з використанням класів мовою ***C#***.

(max: 5 балів)

Теоретичні відомості:

1. Клас як модуль і клас як тип даних в C#.

Програма на мові C# складається з класів, усередині яких описують методи і дані. Змінні, описані безпосередньо усередині класу, називаються *полями класу*. Їм автоматично призначається так зване "значення за замовчанням" — як правило, це 0 відповідного типу. Змінні, описані усередині методу класу, називаються *локальними змінними*. Їх ініціалізація покладається на програміста.

В C# клас грає 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, що будується за модульним принципом, складається з класів, що є основним видом модуля.

У мові C# дозволяється оголосити клас, який не розглядається як тип даних, і в якого зберігається єдина роль – роль модуля. Такий клас оголошується з модифікатором **static**. В цьому класі можуть бути задані константи, лише статичні поля і лише статичні методи. У нього немає конструкторів в звичайному сенсі цього слова, що дозволяють створювати об'єкти – екземпляри класу.

Статичним є клас, всі методи якого є статичними. Статичні класи не потребують створення об'єкту оператором **new**.

Статичний клас не може розглядатися як тип даних.

Приклад – клас **Program** консольної програми зі статичним методом **main.**

Практично велику систему, що створюється колективом розробників, без розділення системи на модулі побудувати неможливо.

Модульність побудови – основний засіб боротьби із складністю системи.

Клас – це тип даних, який задає реалізацію деякої абстракції даних, характерної для проблемної області, для якої створюється програмна система.

Склад класу як типу даних визначається не архітектурними міркуваннями, а тією абстракцією даних, яку повинен реалізувати клас.

Об'єкти класу як *типу даних* є динамічними і створюються за допомогою оператора **new**.

В C# оголошення класів вводяться ключовим словом *class*.

Клас є основою для створення об'єктів. В класі визначаються дані і код, який працює з цими даними.

Методи і змінні, що складають клас, називаються *членами класу*. При визначенні класу оголошуються змінні, які він містить і код, який працює з цими змінними. Безпосередньо ініціалізація змінних в об'єкті (змінних екземпляра) виконується в *конструкторі*.

Об'єкти є *екземплярами* класу. Це динамічні об'єкти, вони створюються при виконанні програми в області видимості і знищуються після виходу з неї.

Формально клас описується таким чином:

Синтаксис опису класу:

```
тип доступу class ім'я_класу {
тип_доступу тип ім'я змінної1;
тип_доступу тип ім'я змінної2;
…
тип_доступу тип_результату
ім'я _методу1(список_параметрів) {тіло_методу}
}
```

де тип_доступу визначає область видимості класу.

Для класів визначено такі модифікатори доступу:

public - клас доступний для інших компонент;
internal - клас видимий в середині цього компонента (збірки).
Для членів класу (даних і методів) визначені такі модифікатори доступу:
public - члени класу доступні за межами даного класу;
internal - члени класу доступні в межах однієї збірки;
protected - члени класу доступні усередині даного класу;
private - члени класу доступні тільки для інших членів даного класу.
За замовчанням застосовується модифікатор **internal.**
Членами класу можуть бути:
• Константи (const)
• Поля (field)
• Конструктори (у тому числі без параметрів)
• Деструктори
• Методи
• Властивості (property)
• Індексатори (властивості з параметрами)
• Події (event)
• Вкладені типи.

Звернення до поля класу виконується за допомогою операції доступу (крапка). Праворуч від крапки задається ім'я поля, зліва — ім'я екземпляра для звичайних полів або ім'я класу для статичних. Створення об'єкту класу виконується оператором **new**.

Синтаксис оператора **new**:

```
Ім'я класу  об'єкт = new Ім'я класу();
```

2. Основні принципи ООП і їх реалізація в C#

Основними принципами об'єктно-орієнтованого програмування (ООП) є **абстракція, інкапсуляція, спадкоємство і поліморфізм**. Реалізація цих принципів у різних мовах програмування може дещо відрізнятися.

Розглянемо спочатку визначення цих понять і як вони реалізовані в C# .

Абстракція - опис взаємодії виключно в термінах повідомлень/подій в предметній області. Події обробляються методами класу, визначеними з модифікатором доступу *public*. Поля і методи, які мають модифікатор доступу *public,* утворюють так званий *інтерфейс класу* (не плутати з інтерфейсними класами, які ми розглянемо пізніше!).

Інкапсуляція - здатність приховування реалізації (властивостей і методів усередині класу), тобто можливість доступу до *об'єкту* і маніпулювання ним виключно за допомогою *властивостей* і *методів* класу.

До основних властивостей *інкапсуляції* відносяться наступні:
- сумісне зберігання даних і функцій (тобто *властивостей* і *методів*) усередині *класу*;
- приховування внутрішньої інформації від користувача (що забезпечує більшу безпеку програмі). Забезпечується модифікаторами доступу;
- ізоляція користувача від деталей реалізації (що забезпечує незалежність від моделі обчислень і потенційно дружній інтерфейс програми).

Спадкоємство (наслідування) - можливість породжувати один клас від іншого із збереженням всіх властивостей і методів базового класу і додавання, за потреби, нових властивостей і методів.

На відміну від C++, у мові C# заборонено множинне спадкоємство, тобто, похідний клас (клас-нащадок) може мати тільки один батьківський клас.

Спадкоємство призначене для відображення такої властивості систем, як *ієрархічність*.

Поліморфізм – це концепція, що дозволяє мати різні реалізації для одного і того самого *методу*, які будуть вибиратися залежно від *типу об'єкту*, переданого до методу при його виклику.

В мові C# поліморфізм реалізується через перевантаження методів і операцій.

В *об'єктно-орієнтованому* програмуванні під ***поліморфізмом*** розуміється можливість оперувати *об'єктами*, не володіючи точним знанням їх типів.

3. Методи класу

Методи це процедури і функції, які реалізують необхідні дії над змінними класу. Область видимості методів визначається модифікаторами доступу. За замовчанням застосовується модифікатор *internal***.**

Синтаксис опису методу:

```
<модифікатор-доступу> <тип результату> <ім'я методу> (опис-параметрів)
                                { тіло-методу }
```

Приклад 1

```
void A() {...};
int B(){...};
public void C(){...};
```

Методи A і B є закритими, а метод C - відкритий. Методи A і C реалізовані процедурами, а метод B - функцією, що повертає ціле значення.

Методи можуть мати параметри, які використовуються для обміну інформацією з методом. Параметр є *локальною змінною*, яка при виклику методу набуває значення відповідного аргументу. Зона дії параметра — весь метод.

Наприклад, для того щоб обчислити значення синуса для дійсної величини x, ми передаємо її як аргумент в метод `Sin` класу `Math`, а щоб вивести значення цієї змінної на екран, ми передаємо її в метод `WriteLine` класу `Console`:

```
double x = 0.1;
double y = Math.Sin(x);
Console.WriteLine(x);
```

При цьому метод `Sin` повертає в точку свого виклику дійсне значення синуса, яке призначається змінній y, а метод `WriteLine` нічого не повертає.

Обов'язковим при описі заголовка методу є вказівка типу результату, імені методу і круглих дужок, наявність яких необхідна і в тому випадку, якщо сам список формальних параметрів відсутній. Формально тип результату методу вказується завжди, але значення void однозначно визначає, що метод реалізується процедурою. Тип результату, відмінний від void, указує на функцію.

Увага!

Методи можуть бути описані в будь-якому місці класу. На відміну від C++ , C# не вимагає оголошувати методи до їхнього опису.

4. Параметри методів

При виклику методу виконуються наступні дії:

1. Обчислюються вирази, що стоять на місці аргументів.
2. Виділяється пам'ять під параметри методу відповідно до їх типу.
3. Кожному з параметрів зіставляється відповідний аргумент (аргументи як би накладаються на параметри і заміщають їх).
4. Виконується тіло методу.
5. Якщо метод повертає значення, воно передається в точку виклику; якщо метод має тип void, управління передається на наступний після виклику оператор.

При цьому перевіряється відповідність типів аргументів і параметрів і, при необхідності, виконується їх перетворення. При невідповідності типів видається повідомлення про помилку.

Приклад 2. Передача методу параметрів

```
using System;
namespace ConsoleApplication1
{   class Class1
   {
    static int Max(int a, int b) // метод вибору максимального значення
        {
            if ( a > b ) return a;
            else         return b;
        }
        static void Main()
        {
            int a = 2, b = 4;
            int x = Max( a, b );                // виклик методу Max
            Console.WriteLine( x );             // результат: 4
            short t1 = 3, t2 = 4;
            int y = Max( t1, t2 );              // виклик методу Max
            Console.WriteLine( y );             // результат: 4
            int z = Max( a + t1, t1 / 2 * b );  // виклик методу Max
```

```
            Console.WriteLine( z );              // результат: 5
        }
    }
}
```

Увага!
Головна вимога при передачі параметрів полягає в тому, що аргументи при виклику методу повинні записуватися в тому самому порядку, що і в заголовку методу, і повинне існувати неявне перетворення типу кожного аргументу до типу відповідного параметра. Кількість аргументів повинна відповідати кількості параметрів.

Існують два способи передачі параметрів: *за значенням і за посиланням*.

При передачі за значенням метод отримує копії значень аргументів, і оператори методу працюють з цими копіями. Доступу до значень аргументів у методу немає, а, отже, немає і можливості їх змінити.

При передачі за посиланням (за адресою) метод отримує копії адрес аргументів, він здійснює доступ до елементів пам'яті по цих адресах і може змінювати значення аргументів, модифікуючи параметри.

У C# для обміну даними між викликаючими функціями і функціями, що викликаються, передбачено чотири типи параметрів:

- параметри-значення;
- параметри-посилання — описуються за допомогою ключового слова ref;
- вихідні параметри — описуються за допомогою ключового слова out;
- параметри-масиви — описуються за допомогою ключового слова params.

Ключове слово передує опису типу параметра. Якщо воно опущене, параметр вважається параметром-значенням. Параметр-масив може бути лише один і повинен бути останнім в списку, наприклад:

```
public int Calculate( int a, ref int b, out int c, params int[] d ) …
```

5. Використання показчика this в тілі класу

Кожен об'єкт містить свій екземпляр полів класу. Методи знаходяться в пам'яті в одному екземплярі і використовуються всіма об'єктами спільно, тому необхідно забезпечити роботу методів нестатичних екземплярів з полями саме того об'єкту, для якого вони були викликані. Для цього в будь-який нестатичний метод автоматично передається прихований параметр this, в якому зберігається посилання на поточний екземпляр об’єкта, що викликав функцію.

Це посилання є прихованим показчиком на кожний нестатичний метод класу. Будь-який метод може використовувати ключове слово this для доступу до інших нестатичних методів і змінних цього об’єкта.

У явному вигляді параметр this застосовується для того, щоб повернути з методу посилання на об'єкт, а також для ідентифікації поля в разі, якщо його ім'я збігається з іменем параметра методу, наприклад:

Приклад 3. Показчик this в тілі класу

```
class Demo
{
    double y;
    public Demo T()         // метод повертає посилання на екземпляр
    {
        return this;
    }
    public void Sety( double y )
    {
        this.y = y;     // полю y призначається значення параметра y
    }
}
```

6. Конструктори. Типи конструкторів

Конструктор – це спеціальний метод, який виділяє пам'ять під об'єкт і ініціалізує виділену область. Його назва співпадає з іменем класу. Особливість конструктора – він не повертає результатів.

Синтаксис конструктора:

```
ім'я_класу(список_параметрів) {тіло_конструктора}
```

Деструктор – метод, що викликається **автоматично** при знищенні об'єкта класу (безпосередньо перед "збиранням сміття"). Деструктор не має параметрів і не повертає результат.

Синтаксис деструктора:

```
~ім'я_класу() {тіло деструктора}
```

При створенні об'єкту відбувається виклик відповідного конструктора класу.

```
ім'я класу ім'я об'єкту = new ім'я класу();
```

Приклад

```
Variables var = new Variables();
```

Оператор **new** виділяє пам'ять, а `Variables()` ініціалізує її.

Властивості конструктора:

- Конструктор не повертає значення (навіть типу void).
- Конструктор може бути з параметрами і без параметрів.
- В класі можуть бути визначені декілька конструкторів з різними списками параметрів.
- Якщо конструктор відсутній, то він створюється автоматично (конструктор за замовчуванням). Такий конструктор не має параметрів.
- Якщо в класі визначений хоча б один конструктор, конструктор за замовчуванням не створюється.

До цих пір ми задавали початкові значення полів класу при описі класу. Це зручно у тому випадку, коли для всіх екземплярів класу початкові значення деякого поля однакові. Якщо ж при створенні об'єктів потрібно призначати полю різні значення, це слід робити в конструкторі. У прикладі 4 в клас `Demo` доданий конструктор, а поля зроблені закритими.

Приклад 4

```
using System;
namespace ConsoleApplication1
{
    class Demo
    {
        public Demo( int a, double y )      // конструктор з параметрами
        {
            this.a = a;
            this.y = y;
        }
        public double Gety()                 // метод отримання поля y
        {
            return y;
        }
        int a;
        double y;
    }
    class Class1
    {   static void Main()
        {
            Demo a = new Demo( 300, 0.002 );      // виклик конструктора
            Console.WriteLine( a.Gety() );           // результат: 0,002
            Demo b = new Demo( 1, 5.71 );        // виклик конструктора
            Console.WriteLine( b.Gety() );            // результат: 5,71
        }
    }
}
```

Часто буває зручно задати в класі декілька конструкторів, аби забезпечити можливість ініціалізації об'єктів різними способами. Всі конструктори повинні мати різні сигнатури (кількість і типи параметрів).

Якщо один з конструкторів виконує які-небудь дії, а інший повинен робити те ж саме плюс ще що-небудь, зручно викликати перший конструктор з другого. Для цього використовується вже відоме вам ключове слово this в іншому контексті, наприклад:

Приклад 5.

```
class Demo
{
    public Demo( int a )                                  // конструктор 1
```

```
        {
            this.a = a;
        }
        public Demo(int a, double y): this(a) // виклик конструктора 1
        {
            this.y = y;
        }
        ...
    }
```

Конструкція, що знаходиться після двокрапки, називається ініціалізатором.

Всі класи в C# мають спільного предка — клас `object`. Конструктор будь-якого класу, якщо не вказаний ініціалізатор, автоматично викликає конструктор свого предка.

До цих пір йшлося про "звичайні" конструктори, або конструктори екземпляра. Існує другий тип конструкторів — статичні конструктори, або конструктори класу. Конструктор екземпляра ініціалізує дані екземпляра, конструктор класу — дані класу.

Статичний конструктор не має параметрів, його не можна викликати явним чином. Система сама визначає момент, в який потрібно його виконати.

Деякі класи містять лише статичні дані і, отже, створювати екземпляри таких об'єктів не має сенсу. У версію 2.0 введена можливість описувати статичний клас, тобто клас з модифікатором static. Екземпляри такого класу створювати заборонено, і крім того, від нього заборонено успадковувати. Всі елементи такого класу повинні явним чином оголошуватися з модифікатором static (константи і вкладені типи класифікуються як статичні елементи автоматично). У прикладі 6 наведено опис статичного класу.

Приклад 6.

```
using System;
namespace ConsoleApplication1
{
    static class D
    {
        static int a = 200;
        static double b = 0.002;

        public static void Print ()
        {
            Console.WriteLine( "a = " + a );
            Console.WriteLine( "b = " + b );
        }
    }

    class Class1
    {   static void Main()
        {
            D.Print();
        }
    }
}
```

Приклад 7.

Розглянемо ще один приклад - клас Animal. В ньому використовується конструктор з параметрами. Жовтим кольором виділено сигнатуру конструктора.

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Inheritance
{
  class Animal
  {
```

```
    public string Name;
    private int Weight;
    protected int Typ;
    public Animal(int W, int T, string N)    //конструктор з параметрами
    {
      Weight = W;
      Typ = T;
      Name = N;
    }
    public int GetWeight() { return Weight; }
    public int GetTyp() { return Typ; }
  }
    class Program
    {
        static void Main(string[] args)
        {
            int weight;
           // конструктор з параметрами
            Animal myAnimal = new Animal(5, 1, "Cat");
            weight=myAnimal.GetWeight();
            Console.WriteLine("Вага = " + weight);
// Console.WriteLine("Тип = " + Typ); //так не можна! тип protected
            Console.WriteLine("Name= " + myAnimal.Name);
            Console.ReadLine();
        }
    }
}
```

7. Властивості get і set

В C# є два спеціальні методи, які називаються *властивостями*.

Властивості призначені для організації доступу до полів класу. Зазвичай властивість пов'язана із закритим полем класу і визначає методи його отримання і призначення.

Синтаксис властивості:

```
[ атрибути ] [ модифікатор ] тип ім'я_властивості
                              {
                    [ get   код_доступу ]
                    [ set   код_доступу ]
                              }
```

Значення модифікаторів доступу для властивостей і методів аналогічні. Частіше за все властивість оголошують із модифікатором доступу `public`. Код доступу є блоками операторів, які виконуються при отриманні (get) або встановленні (set) властивості. Може бути відсутньою або частина get, або set, але не обидві одночасно.

Якщо відсутня частина set, властивість доступна лише для читання (read-only), якщо відсутня частина get, властивість доступна лише для запису (write-only). Починаючи з версії C# 2.0 введена можливість задавати різний рівень доступу для частин get і set.

Приклад 8. Опис властивостей:

```
public class Button: Control
{
   private string caption; //закрите поле, з яким пов'язана властивість
   public string Caption {     // властивість
      get {                          // спосіб отримання  властивості
               return caption;
            }
       set {                        // спосіб призначення  властивості
          if (caption != value) {
             caption = value;
          }
       }
```

```
    }
}
```

Метод запису зазвичай містить дії з перевірки допустимості встановлюваного значення, метод читання може містити, наприклад, підтримку лічильника звернень до поля.

У програмі властивість виглядає як поле класу, наприклад:

```
Button ok = new Button();
ok.Caption = "OK";       // викликається метод призначення  властивості
string s = ok.Caption;     // викликається метод отримання  властивості
```

При зверненні до властивості автоматично викликаються вказані в ньому методи читання і запису.

Синтаксично читання і запис властивості виглядають майже як методи. Метод get повинен містити оператор return. У методі `set` використовується параметр із стандартним ім'ям *`value`*, який містить встановлюване значення.

Приклад 9

В цьому прикладі код містить властивість з іменем `ForeName`, яка ініціалізує поле `foreName`, накладаючи на нього обмеження на довжину.

```
private string foreName;
public string ForeName
{
  get
     { return foreName;  }
}
  set
  {
     if (value.Length >20)
    //деякий код
    else
        foreName=value;
}
}
```

Розглянемо ще один приклад.

Приклад 10.

Клас `Client` містить інформацію про клієнта банку таку як: Прізвище, Дата народження, Дані паспорту, ID-код, номер рахунку. В класі два конструктори. Метод `EditClient()` дозволяє змінювати дані про клієнта. Поля класу зроблені закритими, а доступ до них виконується через властивості get() і set().

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ClientAccount
{
    public class Client
    {
        //поля класу
        private string Name;
        private DateTime BirthDate;
        private string Passport;
        private string ID;
        private string nomAccount;
        public Client()
        {
            //конструктор без параметрів
        }
        public Client (string ClientName, string ClientPassport,
DateTime ClientBirthDate)
        {
          //конструктор з параметрами
            name = ClientName;
            passport = ClientPassport;
            birthdate = ClientBirthDate;
```

```
            }
            public void EditClient(string ClientName, string
ClientPassport, DateTime ClientBirthDate)
            {
              //метод редагування полів класу
                Name = ClientName;
                Passport = ClientPassport;
                birthdate = ClientBirthDate;
            }
        //методи get і set для полів класу
            public string passport
            {
                get
                {
                    return Passport;
                }
                set
                {
                    Passport = value;
                }
            }
            public string name
            {
                get
                {
                    return Name;
                }
                set
                {
                    Name = value;
                }
            }
            public int age
            {
                get
                {
                    int a;
                    a = DateTime.Now.Year - BirthDate.Year;
                    return a;
                }
            }
            public DateTime birthdate
            {
                get
                {
                    return BirthDate;
                }
                set
                {
                    if (DateTime.Now > value)
                        BirthDate = value;
                    else
                throw new Exception("Введено неправильну дату народження");
                }
            }
            public string ID_kod
            {
                get {return ID; }
                set
                {
```

```
                ID = value;
            }
        }
        public string Nom_Account
        {
            get { return nomAccount; }
            set
            {
                nomAccount = value;
            }
        }
    }
```

В методі `Main` ілюструється два способи роботи з цим класом. В першому випадку (закрито коментарями) викликається конструктор без параметрів, а ініціалізація полів `name, passport, birthdate` виконується через властивості get і set.

У другому випадку викликається конструктор з параметрами. Через властивості ініціалізуються поля `ID_kod` та `Nom_Account`.

Приклад 11. Робота з класом Client. Два варіанти

```
    class Program
    {
        static void Main(string[] args)
        {
         //  конструктор без параметрів
         // Client c1 = new Client();
         //ініціалізація полів класу через властивості set.
        //   c1.name = "Вася";
        //   c1.passport = "9002";
        //   c1.birthdate = new DateTime(1987, 08, 03);
        // конструктор з параметрами
   Client c1 = new Client("Вася", "9002", new  DateTime(1987, 08, 03));
             c1.ID_kod = "123456789";
             c1.Nom_Account = "8097";
           Console.WriteLine("Ім'я=" + c1.name);
           Console.WriteLine("Паспорт=" + c1.passport );
           Console.WriteLine("Вік=" + c1.age);
           Console.WriteLine("Іден.код=" + c1.ID_kod);
           Console.WriteLine("Код рахунку=" + c1.Nom_Account);
           Console.ReadKey();
          // редагування полів
         c1.EditClient("Іван", "4567", new DateTime(1987, 09, 01));
           //Відредаговані поля
           Console.WriteLine("Ім’я=" + c1.name);
           Console.WriteLine("Паспорт=" + c1.passport);
           Console.WriteLine("Вік=" + c1.age);
           Console.ReadKey();
        }
    }
}
```

Як вже говорилося в лекції 7, ***поліморфізм*** – це концепція, що дозволяє мати різні реалізації для одного і того ж *методу*, які будуть вибиратися залежно від *типу об'єкту*, переданого до методу при його виклику.

Ця концепція в C# реалізується як перевантаження (методів, операцій).

8. Перевантаження методів

Часто буває зручно, аби методи, що реалізують один і той самий алгоритм для різних типів даних, мали одне і те саме ім'я. Використання декількох методів з одним і тим самим іменем, але різними типами параметрів називається *перевантаженням методів.*

Компілятор визначає, який саме метод потрібно викликати за типом фактичних параметрів. Наприклад, нижче наведено декілька реалізацій метода max, який повертає найбільше значення для різних типів і кількості параметрів.

```
...
// Повертає найбільше з двох цілих:
int max( int a, int b )
// Повертає найбільше з трьох цілих:
int max( int a, int b, int з )
// Повертає найбільше першого параметра і довжини другого:
int max ( int a, string b )
// Повертає найбільше другого параметра і довжини першого:
int max ( string b, int a )
...
Console.WriteLine( max( 1, 2 ));
Console.WriteLine( max( 1, 2, 3 ));
Console.WriteLine( max( 1, "2" ));
Console.WriteLine( max( "1", 2 ));
...
```

При виклику методу max компілятор вибирає варіант методу, який відповідає типу аргументів методу.

Якщо точної відповідності не знайдено, виконуються неявні перетворення типів відповідно до загальних правил. Якщо перетворення неможливе, видається повідомлення про помилку. Якщо відповідність на одному і тому ж етапі може бути отримана більш ніж одним способом, вибирається варіант, що містить меншу кількість і довжину перетворень. Якщо існує декілька варіантів, з яких неможливо вибрати кращий, видається повідомлення про помилку.

Перевантажені методи мають одне **ім'я**, але повинні розрізнятися параметрами, точніше їх типами і кількістю.

Перевантаження широко використовується в класах бібліотеки .NET. Наприклад, в стандартному класі `Console` метод `WriteLine` перевантажений 19 разів для виведення величин різних типів.

Перевантажуватися можуть як конструктори, так і звичайні методи класу. Наприклад, клас `Random` має перевантажені конструктори:

Перший конструктор викликається без параметрів.

```
public Random();
```

Інший конструктор з параметром:

```
public Random (int);
```

Перевантажений метод `public int Next();` при кожному виклику повертає позитивне ціле число, рівномірно розподілене в деякому діапазоні. Діапазон задається параметрами методу. Три реалізації методу відрізняються набором параметрів:

`public int Next ()` - метод без параметрів видає цілі позитивні числа у всьому позитивному діапазоні типу int;

`public int Next (int max)` - видає цілі позитивні числа в діапазоні [0,max];

`public int Next (int min, int max)` - видає цілі позитивні числа в діапазоні [min,max].

Метод `public double NextDouble ()` має одну реалізацію. При кожному виклику цього методу видається нове випадкове число, рівномірно розподілене в інтервалі [0,1).

9. Методи зі змінною кількістю аргументів

Інколи буває зручно створити метод, в який можна передавати різну кількість аргументів. Мова C# надає таку можливість за допомогою ключового слова *params*. Параметр, помічений цим ключовим словом, розміщується в списку параметрів останнім і позначає масив заданого типу невизначеної довжини, наприклад:

```
public int Calculate( int a, out int c, params int[] d ).
```

У цей метод можна передати три і більше параметрів. Усередині методу до параметрів, починаючи з третього, звертаються як до звичайних елементів масиву. Кількість елементів масиву отримують за допомогою його властивості `Length`. Як приклад розглянемо метод обчислення середнього значення елементів масиву.

Приклад 12

```
using System;
namespace ConsoleApplication1
{    class Class1
    {
        public static double Average( params int[] a )
        {
```

```
            if (a.Length == 0)
               throw new Exception("Недостатньо аргументів в методі" );
            double av = 0;
            foreach (int elem in a) av += elem;
            return av / a.Length;
        }
        static void Main()
        {
           try
            {
                int[] a = { 10, 20, 30 };
                Console.WriteLine(Average(a));            // 1
                int[] b = { -11, -4, 12, 14, 32, -1, 28 };
                Console.WriteLine(Average( b ));           // 2
                short z = 1, e = 12;
                byte v = 100;
                Console.WriteLine(Average(z, e, v));      // 3
                Console.WriteLine(Average());              // 4
            }
            catch( Exception e )
            {
                Console.WriteLine(e.Message );
                return;
            }
        }
    }
}
```

Результат роботи програми:
10
20
40
Недостатньо аргументів в методі

Параметр-масив може бути лише один і повинен розташовуватися останнім в списку. Відповідні йому аргументи повинні мати типи, для яких можливе неявне перетворення до типа масиву.

10. Індексатори

Індексатор є різновидом властивості. Якщо в класі є *закрите (private)* поле, яке є масивом, то за допомогою індексатора можна звернутися до елементу цього масиву, використовуючи ім'я об'єкту і номер елементу масиву в квадратних дужках. Іншими словами, індексатор — це такий "розумний" індекс для об'єктів.

Синтаксис індексатора аналогічний синтаксису властивості:

```
атрибути модифікатори  тип this [ список_параметрів ]
                              {
                        get код_доступу
                        set код_доступу
                              }
```

Індексатори найчастіше оголошуються із модифікатором `public`, оскільки вони входять в інтерфейс об'єкту. Атрибути і специфікатори можуть бути відсутніми.

Код доступу є блоками операторів, які виконуються при отриманні (get) або встановленні значення (set) елементу масиву. Може бути відсутньою або частина get, або set, але не обидві одночасно. Якщо відсутня частина set, індексатор доступний лише для читання (read-only), якщо відсутня частина get, індексатор доступний лише для запису (write-only).

Список параметрів містить одне або декілька описів індексів, по яких виконується доступ до елементу. Найчастіше використовується один індекс цілого типу.

Індексатори в основному застосовуються для створення спеціалізованих масивів, на роботу з якими накладаються які-небудь обмеження.

11. Перевантаження операцій

Клас — це тип даних, тому програміст може визначити для нього власні операції. Ви вже стикалися з перевантаженням (тобто перевизначенням) операцій: наприклад, знак "+" для арифметичних типів означає додавання, а для рядків — склеювання. Операції перевантажують в основному для класів, за допомогою яких задають які-небудь математичні поняття, тобто тоді, коли знаки операцій мають загальноприйняту семантику.

11.1.Унарні операції

Можна визначати в класі наступні унарні операції:

```
+      -      !      ~      ++      --      true      false
```

Синтаксис визначення унарної операції:

```
тип operator  унарна_операція ( параметр )
```

Приклади заголовків унарних операцій:

```
public static int operator +( MyObject m )
public static MyObject operator --( MyObject m )
public static bool operator true( MyObject m )
```

Операції не повинні змінювати значення операнду, який передається ним. Операція, що повертає величину типа класу, для якого вона визначається, повинна створити новий об'єкт цього класу, виконати з ним необхідні дії і передати його як результат.

11.2. Бінарні операції

Можна визначати в класі наступні бінарні операції:

```
+    -    *    /    %    &    |    ^    <<    >>    ==    !=    >    <    >=    <=
```

Синтаксис визначення бінарної операції:

```
тип operator бінарна_операція (параметр1, параметр2)
```

Приклади заголовків бінарних операцій:

```
public static MyObject operator +  ( MyObject m1, MyObject m2 )
public static bool     operator == ( MyObject m1, MyObject m2 )
```

Хоч б один параметр, який передається в операцію, повинен мати тип класу, для якого вона визначається. Операція може повертати величину будь-якого типу.

Операції `== і !=, > і <, >= і <=` визначаються лише парами і зазвичай повертають булеве значення. Найчастіше в класі визначають операції порівняння на рівність і нерівність для того, щоб забезпечити порівняння об'єктів, а не їх посилань, як визначено за замовчанням для посилкових типів.

11.3. Операції перетворення типу

Операції перетворення типу забезпечують можливість явного і неявного перетворення між типами даних користувача. Синтаксис визначення операції перетворення типу:

```
implicit operator тип ( параметр )        // неявне перетворення
explicit operator тип ( параметр )        // явне перетворення
```

Ці операції виконують перетворення з типу параметра до типу, вказаного в заголовку операції. Одним з цих типів має бути клас, для якого визначається операція. Таким чином, операції виконують перетворення або типу класу до іншого типу, або навпаки. Перетворювані типи не повинні бути зв'язані відношеннями спадкоємства.

- при використанні об'єкту у виразі, що містить змінні цільового типу;
- при передачі об'єкту в метод на місце параметра цільового типа;
- при явному приведенні типу.

Явне перетворення виконується при використанні операції приведення типу.

Всі операції класу повинні мати різні сигнатури. На відміну від інших видів методів, для операцій перетворення тип значення, яке повертається включається в сигнатуру, інакше не можна було б визначати варіанти перетворення цього типу в інші. Ключові слова `implicit` і `explicit` в сигнатуру не включаються, отже, для одного і того ж перетворення не можна визначити одночасно явну і неявну версію.

Неявне перетворення слід визначати так, щоб при його виконанні не виникала втрата точності і не генерувалися виключення. Якщо ці ситуації можливі, перетворення слід описати як явне.

12. Приклад класу з перевантаженими методами і операціями

Розглянемо рішення наступної задачі.

У текстовому файлі зберігається база відділу кадрів підприємства. На підприємстві 100 співробітників. Кожен рядок файлу містить запис про одного співробітника. Формат запису: прізвище і

ініціали (30 поз., прізвище повинне починатися з першої позиції), рік народження (5 поз.), оклад (10 поз.). Написати програму, яка по заданому прізвищу виводить на екран відомості про співробітника, підраховуючи середній оклад всіх відібраних співробітників.

Для вирішення цієї задачі створимо клас **Person** і організуємо з екземплярів цього класу масив. При описі класу корисно задатися наступним питанням: які обов'язки мають бути покладені на цей клас? Вочевидь, що перший обов'язок — зберігання відомостей про співробітника. Аби скористатися цими відомостями, клієнт (тобто код, що використовує клас) повинен мати можливість отримати ці відомості, змінити їх і вивести на екран. Окрім цього, для пошуку співробітника бажано мати можливість порівнювати його ім'я із заданим.

Поля класу зробимо відповідно до принципу інкапсуляції закритими, а доступ до них організуємо за допомогою властивостей. Це дозволить контролювати процес занесення даних в поля. Для ідентифікації спроби введення невірних даних скористаємося механізмом виключень. Щоб зробити клас більш універсальним, введемо в нього можливість збільшувати і зменшувати оклад співробітника за допомогою перевантажених операцій класу.

Вхідні дані. База співробітників знаходиться в текстовому файлі. Перш за все треба вирішити, чи зберігати в оперативній пам'яті одночасно всю інформацію з файлу чи можна обійтися буфером на один рядок. Якби відомості про співробітника запитувалися одноразово, можна було б зупинитися на другому варіанті, але оскільки пошук по базі виконуватиметься більше одного разу, всю інформацію бажано зберігати в оперативній пам'яті, оскільки багатократне читання з файлу нераціонально.

Максимальна кількість рядків файлу за умовою завдання обмежена, тому можна виділити для їх зберігання масив з 100 елементів. Кожен елемент масиву міститиме відомості про одного співробітника, організовані у вигляді класу.

У програму потрібно також вводити прізвища потрібних співробітників. Чергове прізвище, що вводиться, зберігатимемо в стандартному рядку типу string.

Результати. В результаті роботи програми потрібно вивести на екран необхідні елементи результуючого масиву. Оскільки ці результати є вибіркою з вхідних даних, додаткова пам'ять для них не відводиться. Крім того, необхідно підрахувати середній оклад для знайдених співробітників. Для цього необхідна змінна дійсного типу.

Проміжні величини. Для пошуку середнього окладу необхідно підрахувати кількість співробітників, для яких виводилися відомості. Заведемо для цього змінну цілого типа.

II. Алгоритм рішення задачі:

1. Ввести з файлу в масив відомості про співробітників.
2. Організувати цикл виведення відомостей про співробітника:
   - ввести з клавіатури прізвище;
   - виконати пошук співробітника в масиві;
   - збільшити сумарний оклад і лічильник кількості співробітників;
   - вивести відомості про співробітника або повідомлення про їх відсутність;
3. Вивести середній оклад.

Необхідно вирішити, яким чином виконувати вихід з циклу виведення відомостей про співробітників. Умовимося, що для виходу з циклу замість прізвища слід просто натиснути клавішу Enter. Текст програми наведений в прикладі 13.

Приклад 13

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace Person1
{
   class Person
   {                                  // 1
      const int l_name = 30;
      string name;
      int birth_year;
      double pay;
      public Person()          // конструктор без параметрів
      {
         name = "Anonimous";
         birth_year = 0;
```

```
            pay = 0;
        }
        public Person(string s)                    // 2 конструктор з параметром
        {
            name = s.Substring(0, l_name);
            birth_year = Convert.ToInt32(s.Substring(l_name, 4));
            pay = Convert.ToDouble(s.Substring(l_name + 4));
            if (birth_year < 0) throw new FormatException();
            if (pay < 0) throw new FormatException();
        }
        public override string ToString()        // 3 перевантажений метод
        {
            return string.Format("Name: {0,30} birth: {1} pay: {2:F2}",name, birth_year, pay);
        }
        public int Compare(string name) // порівняння прізвища
        {
            return (string.Compare(this.name, 0,name + " ", 0, name.Length +
1,StringComparison.OrdinalIgnoreCase));
        }
        // --------------------  властивості класу -------------------------
        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        public int Birth_year
        {
            get { return birth_year; }
            set
            {
                if (value > 0) birth_year = value;
                else throw new FormatException();
            }
        }
        public double Pay
        {
            get { return pay; }
            set
            {
                if (value > 0) pay = value;
                else throw new FormatException();
            }
        }
        // ------------------  операції класу --------------------------
        public static double operator +(Person pers, double a)
        {
            pers.pay += a;
            return pers.pay;
        }
        public static double operator +(double a, Person pers)
        {
            pers.pay += a;
            return pers.pay;
        }
        public static double operator -(Person pers, double a)
        {
            pers.pay -= a;
            if (pers.pay < 0) throw new FormatException();
            return pers.pay;
```

```
    }
};


class Program
{
    static void Main(string[] args)
    {
        Person[] dbase = new Person[100];
        int n = 0;
        try
        {
            StreamReader f = new StreamReader("dbase.txt");    // 4
            string s;
            int i = 0;
            while ((s = f.ReadLine()) != null)                 // 5
            {
                dbase[i] = new Person(s);
                Console.WriteLine(dbase[i]);
                ++i;
            }
            n = i - 1;
            f.Close();
        }
        catch (FileNotFoundException e)
        {
            Console.WriteLine(e.Message);
            Console.WriteLine("Перевірте правильність імені і шляху до файлу!");
            return;
        }
        catch (IndexOutOfRangeException)
        {
            Console.WriteLine("Дуже великий файл!");
            return;
        }
        catch (FormatException)
        {
            Console.WriteLine("Недопустима дата народження або оклад");
            return;
        }
        catch (Exception e)
        {
            Console.WriteLine("Помилка: " + e.Message);
            return;
        }
        int n_pers = 0;
        double mean_pay = 0;
        Console.WriteLine("Введіть прізвище співробітника");
        string name;
        while ((name = Console.ReadLine()) != "")          // 6
        {
            bool not_found = true;
            for ( int k = 0; k <= n; ++k )
            {
                Person pers = dbase[k];
                if (pers.Compare(name) == 0)
                {
                    Console.WriteLine(pers);
                    ++n_pers; mean_pay += pers.Pay;
```

```
                    not_found = false;
                }
            }
            if (not_found) Console.WriteLine("Такого співробітника немає");
            Console.WriteLine(
          "Введіть прізвище співробітника або Enter для завершення");
        }
        if (n_pers > 0)
                        Console.WriteLine("Середній оклад: {0:F2}", mean_pay / n_pers);
        Console.ReadKey();
      }
    }
}
```

В операторі 1 описаний клас **Person** для зберігання інформації про одного співробітника. Окрім полів даних, в ньому заданий конструктор, що виконує заповнення полів (оператор 2), перевизначений метод перетворення в рядок (оператор 3), що дозволяє виводити екземпляр об'єкту на екран за допомогою методу `WriteLine` класу `Console`, а також описаний метод `Compare`, в якому прізвище співробітника порівнюється із заданим через параметр.

Порівняння прізвища виконується за допомогою одного з варіантів методу `Compare` класу `string`, який дозволяє порівнювати підрядки без врахування регістра. Після заданого прізвища доданий пропуск, оскільки якщо пропуску немає, то потрібне прізвище є частиною іншого, і цей рядок нам не підходить.

Властивості і операції класу дозволяють виконувати введення і редагування окладу і року народження.

Вхідний файл слід створити до першого запуску програми відповідно до формату, заданого в умові завдання. Можна використовувати будь-який текстовий редактор, що підтримує кодування Unicode (наприклад, Блокнот).

У операторі 4 описаний екземпляр стандартного класу символьного потоку. Цикл 5 виконує порядкове прочитування з файлу в рядок s і заповнення чергового елементу масиву dbase. У операторі 6 організується цикл перегляду масиву. Є видимими лише заповнені при введенні елементи.

Індивідуальні завдання:

(Для виконання індивідуальних завдань № варінта є порядковим номером прізвища студента в списку групи. Усі проекти та тести (якщо передбачено завданням) завантажити у власні репозиторії на Git Hub та виконати їх збірку/build у середовищі Travis CI. Посилання на репозиторії проектів вказати у звіті та обов'язково долучати скріни успішної збірки в Travis CI.)

1. Створити клас та розробити консольний застосунок, який реалізує вказані дії. Також розробити модульний тест для перевірки коректності роботи методу. Спільне для всіх варіантів.

(1 бал)

Розробити консольний застосунок для роботи з класом Student з полями: name, LastName, Group, Year, Adress, Passport, Age, Telephon, Rating. Під рейтингом розуміється середній бал за 100-ою системою. Всі поля класу зробити закритими, а доступ до них реалізувати через get і set. Реалізувати конструктор без параметрів, а ініціалізацію полів виконати через властивості. Реалізувати метод `StudentRating(int R)`. Метод повинен виводити відповідне текстове повідомлення, в залежності від рейтингу: якщо рейтинг 90 і вище - повідомлення «Вітаємо відмінника», для рейтингу від 75 балів – «можна вчитися краще», для рейтингу менше 75 балів – «Варто більше уваги приділяти навчанню»

Увага! У тесті спочатку створюється екземпляр класу, а потім виконується перевірка очікуваного результату.

2. Розробити консольний застосунок для роботи з класом згідно свого варіанту. Створити не менше 10 записів для виконання операцій, вказаних у завданні. Для ідентифікації спроби введення з клавіатури некоректних даних описати виключення.

(2 бала)

1. Описати клас для бази зданих з інформацією про книги у формі: автор, назва книги, видавництво, рік видання, кількість штук. Вивести масив на екран у формі таблиці, згрупувавши книги з однаковим роком видання. Порахувати загальну кількість книг для кожного року видання та вивести цю інформацію у вигляді таблиці на екран.
2. Описати клас для бази зданих з інформацією про книги у формі: автор, назва книги, видавництво, рік видання. Вивести дані про книги з програмування (перевіряти, чи є частиною назви книги слово «програмування» з малої або великої літери) у порядку спадання років видань.
3. Описати клас для бази зданих з інформацією про конфігурацію комп'ютера з полями: тип процесора, тактова частота, обсяг оперативної пам'яті, обсяг дискової пам'яті, характеристика монітора та ін. Відсортувати записи по типу процесора та вивести на екран у формі таблиці. Визначити комп'ютери з найбільшим обсягом оперативної і дискової пам'яті.
4. Описати клас для бази зданих з інформацією про результати роботи підприємства протягом року у форматі: місяць, план випуску продукції, фактичний випуск продукції. изначити назви місяців з недовиконанням плану випуску продукції та вивести цю інформацію на екран у вигляді таблиці.
5. Описати клас для бази зданих з інформацією про результати роботи підприємства впродовж року з полями: місяць, план випуску продукції, фактичний випуск продукції. Відсортувати масив записів у порядку зростання відсотку виконання плану та вивести його на екран у формі таблиці. Визначити місяці з найбільшим та найменшим відсотком виконання плану.
6. Описати клас для бази зданих з інформацією про рейтинг студентів однієї групи: прізвище студента, № залікової книжки, рейтинг у 100 бальній шкалі. Впорядкувати записи по спаданню рейтингу та вивести його на екран у формі таблиці. Обчислити середній рейтинг групи та кількість студентів з рейтингом нижче середнього.
7. Описати клас для бази зданих з інформацією про студентів з полями: прізвище, дата народження, місце народження. Дата народження задається у вигляді ДД:ММ:РР. Відсортувати записи за зростанням дат народження та вивести його на екран у формі таблиці. Якщо є студенти з однаковою датою народження (співпадають день та місяць), то вивести записи про них окремо в таблиці «двійнята».
8. Описати клас для бази зданих з інформацією про успішність групи студентів з полями: прізвище та ім'я, № залікової книжки, оцінки за 100 бальною шкалою з п'яти предметів. Впорядкувати записи у порядку зростання зростання середнього балу і вивести їх на екран у формі таблиці. Визначити відсоток студентів, що мають незадовільні оцінки.
9. Описати клас для бази зданих з інформацією про успішність групи студентів з полями: прізвище та ім'я, № залікової книжки, оцінки з п'яти предметів за 100 бальною шкалою. Впорядкувати записи у порядку спадання середньої оцінки та вивести їх на екран у формі таблиці. Визначити відсоток студентів, середній бал яких відповідає оцінкам "добре" та "відмінно".
10. Описати клас для бази зданих з інформацією про метеорологічні спостереження протягом місяця у форматі: дата, температура повітря, атмосферний тиск. Впорядкувати записи у порядку спадання температури повітря та вивести його на екран у формі таблиці. Визначити два дні з найбільшим перепадом температури повітря.
11. Описати клас для бази зданих з інформацією про метеорологічні спостереження протягом місяця у форматі: дата, температура повітря, атмосферний тиск. Визначити дні з атмосферним тиском, більшим від середнього начення цього показника за весь період. Результат вивести на екран у формі таблиці.
12. Описати клас для бази зданих з інформацією про метеорологічні спостереження протягом року у форматі: дата (ДД:ММ), температура повітря. Знайти середню температуру повітря кожного місяця та вивести на екран таблицю: місяць, середня температура. Визначити назву місяця з найбільшою середньою температурою повітря.
13. Описати клас для бази зданих з інформацією про метеорологічні спостереження протягом місяця у форматі: дата, температура, атмосферний тиск. Впорядкувати дані у порядку зростання тиску та вивести результати на екран у формі таблиці. Визначити два дні з найбільшим перепадом тиску.
14. Описати клас для бази зданих з інформацією про метеорологічні спостереження протягом місяця у форматі: дата, температура, тиск. Визначити дні, температура яких є більшою від середнього значення температури. Результат вивести на екран у формі таблиці.
15. Описати клас для бази зданих з інформацією про власників авто з полями: марка автомобіля, номер автомобіля, колір, дані про власника. Вивести на екран у формі таблиці дані про власників автомобілів заданного кольору та заданої марки.

16. Описати клас для бази зданих з інформацією про успішність з полями: прізвище студента, оцінки з п'яти предметів. Знайти середній бал по кожному предмету. Вивести результати на екран у формі таблиці.

17. Описати два класи для бази зданих з інформацією про виробництво. Перший містить поля: назва виробу, вартість одиниці виробу. Другий про результати виробництва за звітний період: дата, назва виробу, кількість. Дані про один виріб можуть повторюватися в різні дати. Обчислити загальну вартість виготовленої за звітний період продукції за кожним видом виробів. Відсортувати утворений масив по зростанню вартості виробів та вивести його на екран у формі таблиці.

18. Описати клас для бази зданих з інформацією про автомобілі з полями: марка, колір, номер, рік випуску, дані про власника. Відсортувати масив даних по марках автомобілів та вивести його на екран у формі таблиці. Визначити кількість різних кольорів кожної марки.

19. Описати клас для бази зданих з інформацією про автомобілі з полями: марка, колір, номер, рік випуску, дані про власника. Визначити кількість автомобілів кожної марки та вивести на екран у формі таблиці впорядкувавши по спаданню кількості автомобілів.

20. Описати клас для бази зданих з інформацією про дні народження декількох студентів з полями: дата, прізвище та ініціали. Дата задається у форматі ДД:ММ:РР. З клавіатури ввести поточну дату. Визначити студента з найближчим днем народження. Вивести на екран дані про студентів, дні народження яких ще будуть у поточному році (відлік вести від поточної дати).

21. Описати клас для бази зданих з інформацією про про клієнтів банку з полями: № рахунку, прізвище та ім'я, сума вкладу, дата проведення операції. Визначити клієнтів банку з найбільшою кількістю банківських операцій та вивести дані про них на екран у формі таблиці.

22. Описати клас для бази зданих з інформацією про клієнтів банку з полями: дата проведення операції прізвище та ім'я, № рахунку, сума безготівкового отримання/переведення, отримано/видано готівкою, залишок вкладу. Вивести на екран у формі таблиці дані про клієнтів банку, які на протязі заданого періоду часу мають найбільшу суму безготівкового отримання коштів на рахунок.

23. Описати клас для бази зданих з інформацією про клієнтів банку з полями: дата проведення операції прізвище та ім'я, № рахунку, сума безготівкового отримання/переведення, отримано/видано готівкою, залишок вкладу. Вивести на екран у формі таблиці дані про клієнтів банку, які на протязі заданого періоду часу виконали безготівкове переведення на загальну суму, яка перевищує задане користувачем граничне значення.

24. Описати клас для бази зданих з інформацією про книги бібліотеки з полями: автор, назва книги, видавництво, рік видання. Відсортувати масив за прізвищами авторів та вивести на екран у формі таблиці. Підрахувати кількість книг від кожного видавництва.

25. Описати клас для бази зданих з інформацією про книги бібліотеки з полями: автор, назва книги, видавництво, рік видання, кількість штук. Дані про книги, видані після 2000 року вивести на екран у формі таблиці, порахувати загальну кількість таких книг.

3. Розробити консольний застосунок для роботи з базою даних, що зберігається у текстовому файлі (початковий масив не менше 10 записів). Структура бази даних описується класом згідно вашого варіанта. Передбачити роботу з довільною кількістю записів. Для ідентифікації спроби введення з клавіатури некоректних даних описати виключення. Всі поля класу зробити закритими, а доступ до них реалізувати через `get`і `set`. Реалізувати конструктори з параметрами та без параметрів, а ініціалізацію полів виконати через властивості. Реалізувати методи для:

– додавання записів;

– редагування записів;

– знищення записів;

– виведення інформації з файла на екран;

– пошук потрібної інформації за конкретною ознакою (поле Параметр пошуку);

– сортування за різними полями (поле Параметр сортування).

Меню програми реалізувати по натисненню на певні клавіші: наприклад, Enter – вихід, п - пошук, р – редагування тощо.

(3 бала)

№ варіанта	Клас	Поля	Параметр сортування	Параметр пошуку
1.	Бібліотека	Інвентарний номер, автор,	Рік видання	Автор

№ варіанта	Клас	Поля	Параметр сортування	Параметр пошуку
		назва, кількість сторінок, рік видання		
2.	Телефонний довідник	Прізвище, ім'я, по батькові, адреса, телефон.	Телефон	Прізвище
3.	Розклад руху літаків	Номер рейсу, тип літака, напрямок руху, періодичність вильоту.	Номер рейсу	Тип літака
4.	Колекція компакт-дисків	Інвентарний номер, назва альбому, об'єм диску, тип, дата запису.	Дата запису	Назва альбому
5.	Записна книжка	Прізвище, ім'я, асса м адреса, телефон, електронна пошта.	Прізвище	Електронна пошта
6.	Предметний покажчик	Слово; номера сторінок, де це слово зустрічається; кількість цих слів на даній сторінці	Номер сторінки	Слово
7.	Розклад пар	Номер пари, день тижня, предмет, прізвище викладача, форма заняття.	День тижня	Прізвище викладача
8.	Список файлів	ім'я файла, розширення, розмір, дата створення, атрибути.	Розширен-ня	Дата створення
9.	Архів програмного забезпечення	Назва програми, операційна система, розмір програми, дата запису	Назва програми	Операційна система
10.	Рахунки банку	Прізвище, ім'я, дата останньої операції, сума вкладу	Сума вкладу	Дата операції
11.	Користувачі локальної мережі	Прізвище, група, обліковий запис, тип облікового запису.	Тип облікового запису	Прізвище
12.	Камера схову	Прізвище, дата здачі, термін зберігання, інвентарний номер та назва	Інвентарний номер	Дата здачі

№ варіанта	Клас	Поля	Параметр сортування	Параметр пошуку
		предмета		
13.	Склад товарів	Інвентарний номер, назва товару, вага, ціна, кількість	Вага	Назва товару
14.	Каса продажу квитків	Назва пункту, час відправлення, дата відправлення, час прибуття, дата прибуття, ціна квитка	Час від-правлення	Назва пункту
15.	Успішність студентів	Прізвище, номер групи, оцінки з трьох предметів	Прізвище	Номер групи
16	Асортимент виробів	Номер виробу, назва, вага, собівартість виробництва, ціна продажу	Назва	Ціна продажу
17	Метеорологічні дані	Дата, місто, атмосферний тиск, температура повітря, швидкість вітру	Дата	Місто
18	Склад автозапчастин	Порядковий номер, назва, марка авто, ціна, кількість	Назва	Марка авто
19	Пацієнти сімейного лікаря	Номер карти, Прізвище, ім’я, по-батькові, рік народження, стать	Прізвище	Стать
20	Викладачі університету	Номер посвідчення, Прізвище, ім’я, по-батькові, кафедра, стать, науковий ступінь	Прізвище	Кафедра
21	Власники квартир ОСББ	Прізвище, номер будинку, номер квартири, стать, сума боргу	Прізвище	Сума боргу
22	Колекція музичних треків	Назва пісні, виконавець, альбом, рік випуску, тривалість пісні	Назва пісні	Виконавець
23	Колекція фільмів	Назва, прізвище режисера, рік випуску, кіностудія,	Назва	Рік випуску

№ варіанта	Клас	Поля	Параметр сортування	Параметр пошуку
		тривалість фільму		
24	ЖРЕПи міста	Назва, адреса, прізвище начальника, кількість підзвітних будинків, район міста	Назва	Район міста
25	Абоненти кабельної мережі	Номер договору, Прізвище, ім’я, адреса, телефон	Номер договору	Прізвище