Розділ 2. Об’єктно-орієнтоване проєктування та програмування. SOLID принципи проєктування. Принципи інкапсуляції, успадкування та поліморфізму ООП

Лабораторна робота № 3. Конструктори та аксесори класів, вкладені та часткові класи, використання текстових файлів в C#

Рейтинг лабораторної роботи №3

№ п.п	*Вид діяльності студента*	*Рейтинговий бал*	*Deadline*
1	Написання коду з 9 завдань	0,5*9 = 4,5	21 Березня
2	Захист роботи	0,5	
3	Звіт з роботи	0,5	
Разом за роботу		5,5	

Мета роботи

1. Навчитися створювати власні класи, використовуючи атрибути та методи класів, включаючи:
   1.1. Конструктори класів
   1.2. Властивості (аксесори) класів
   1.3. Вкладення класів
   1.4. Часткові класи і методи
   1.5. Статичні класи
   1.6. Текстові файли
2. Навчитися реалізовувати доступ до відкритих методів та закритих атрибутів класів.

Методичні рекомендації до виконання лабораторної роботи

1. Прочитайте лекцію та матеріал в підручнику.
2. Прочитайте методичні вказівки до лабораторної роботи та виконайте наведені в ній приклади (вони всі працездатні).
3. При вивченні теми лекції і виконанні завдань зверніть увагу на особливості створення конструкторів класу в C#:
   3.1.1. конструктор не повертає значення;
   3.1.2. конструктор може бути з параметрами і без параметрів;
   3.1.3. в класі можуть бути визначені декілька конструкторів з різними списками параметрів;
   3.1.4. якщо конструктор відсутній, він створюється автоматично (конструктор за замовчуванням). Такий конструктор не має параметрів;
   3.1.5. якщо в класі визначений хоча б один конструктор, конструктор за замовчуванням автоматично системою не додається.
4. Також зверніть увагу, що для доступу до закритих полів в C# призначені спеціальні методи-властивості *get* і *set*. Якщо відсутня частина set, властивість доступна лише для читання (*read-only*), якщо відсутня частина get, властивість доступна лише для запису (*write-only*).
5. Для поглибленого вивчення цього матеріалу прочитайте розділи книги [1].

Порядок виконання роботи

1. Створити директорію Lab3, в якій будуть розміщуватися проекти цієї лабораторної роботи.
2. Виконати завдання свого варіанту у вигляді окремих консольних проектів в одному рішенні
3. Можлива реалізація у вигляді одного консольного проекту з окремими файлами для кожного з класів з використанням меню для демонстрації роботи кожного завдання варіанту.
4. Для кожного класу передбачити окремий файл.

Приклади виконання завдань

1. Програма розрахунку рейтингу студента. Клас Student

Варіант 1. Конструктор з параметрами та створення об'єкту класу

Розглянемо клас *Student*, який містить відкриті поля і конструктор з параметри для їх ініціалізації. Клас містить один відкритий метод *public void StudentRating(int R)*, який виводить відповідний текст, в залежності від рейтингу студента. В методі *Main()* виконується тестування класу.

Зверніть увагу, що опис класу розміщується в просторі імен, а не класі *Program*. Опис класу рекомендується розмістити в окремому файлі – модулі класу.

```
using System; //файл Student.cs
namespace ex1lab3
{
    /// <summary>
    /// клас студент з атрибутами та методами
    /// </summary>
    class Student
    {
       public string Name;     //ім'я
       public int Age;         // вік
       public string Role;     // роль
       public string Faculty; //факультет
       public string Group;    //група
       public int Course;      //курс
       public int Rating;      /рейтинг
       /// <summary>
       /// конструктор з параметрами для ініціалізації полів класу
       /// </summary>
       /// <param name="N">ім'я студента</param>
       /// <param name="A">вік</param>
       /// <param name="R">роль</param>
       /// <param name="F">факультет</param>
       /// <param name="G">група</param>
       /// <param name="C">курс</param>
       public Student(string name, int age, string role, string fac, string gr, int course )
        {
            Name = name;
            Age = age;
            Role = role;
            Faculty = fac;
            Group = gr;
            Course = course ;
        }
       /// <summary>
       /// виведення рекомендацій студенту відповідно до його рейтингу
       /// </summary>
       /// <param name="R">рейтинг студента</param>
        public void StudentRating(int rank)
        {
            Rating = rank;
            if (Rating >= 80)
                Console.WriteLine("Привіт відмінникам");
            else
                if (Rating <= 30)
                Console.WriteLine("Треба вчитися краще!");
            else
                Console.WriteLine("Можна вчитися ще краще!");
        }
    }
}
//======= файл Program.cs містить точку входу в програму Main()=======
using System;
namespace ex1lab3
{
    class Program
    {
      static void Main(string[] args)
      {
            //дані рейтингу
 ./ініціалізація полів класу виконується в конструкторі з параметрами
        Student newStudent = new Student("Бака", 20, "студент", "КННІ", "К-01", 3);
        Console.WriteLine("Хто ви?");
        Console.WriteLine("Прізвище = " + newStudent.Name);
        Console.WriteLine("Вік= " + newStudent.Age);
        Console.WriteLine("Роль= " + newStudent.Role);
        Console.WriteLine("Факультет = " + newStudent.Facultet);
```

```
            Console.WriteLine("група= " + newStudent.Group);
            Console.WriteLine("курс= " + newStudent.Course);
            Console.WriteLine("Вкажіть Ваш рейтинг?");
            string r = Console.ReadLine();
            newStudent.Rating = int.Parse(r);
            newStudent.StudentRating(newStudent.Rating);
            Console.ReadLine();
        }
    }
}
```

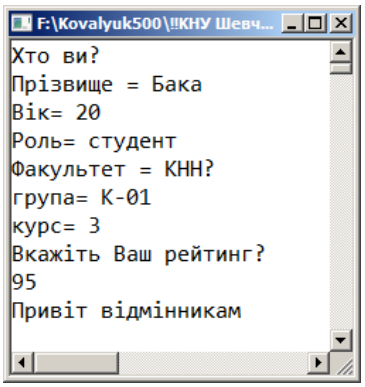

Рисунок 1 – Результат роботи програми 1

Змінимо клас, зробимо поля *Name* і *Age* закритими, а доступ до них реалізуємо через властивості *get* і *set*.

Варіант 2. Використання властивостей *get* і *set* замість присвоєння в конструкторі

```
using System; //файл Student.cs для класу Student
using System.Collections.Generic;
using System.Text;
namespace ex2Lab3
{
    class Student
    {
            //закриті поля класу
        private string name;
        private int age;
            //відкриті поля класу
        public string Role;     // роль
        public string Faculty;
        public string Group;
        public int Course;
        public int Rating;
        /// <summary>
        /// конструктор класу з параметрами
        /// </summary>
        /// <param name="F">факультет</param>
        /// <param name="G">група</param>
        /// <param name="C">курс</param>
        /// <param name="R">роль</param>
        public Student(string fac, string gr, int course, string role)
        {
            //конструктор з параметрами
            Role = role;
            Faculty = fac;
            Group = gr;
            Course = course;
        }
        /// <summary>
        /// властивість для доступу до закритого поля Name через get і set для полів класу
        /// </summary>
        public string Name
        {
            get
            { return Name; }
            set
            { Name = value;}
        }
        /// <summary>
        ///  властивість для доступу до закритого поля Age через get і set для полів класу
        /// </summary>
        public int Age
        {
            get
```

```
            { return age; }
            set
            { age = value; }
        }
        /// <summary>
        /// виведення повідомлень в залежності від значення рейтингу
        /// </summary>
        /// <param name="Rat">рейтинг студента</param>
        public void StudentRating(int rat)
        {
            Rating = rat;
            if (Rating >= 80)
                Console.WriteLine("Привіт відмінникам");
            else
                if (Rating <= 30)
                    Console.WriteLine("Треба вчитися краще!");
                else
                    Console.WriteLine("Можна вчитися ще краще!");
        }
    }
}
//файл Program.cs для класу Program
using System;
namespace ex2Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
             //дані рейтингу
             //ініціалізація полів класу виконується в конструкторі з параметрами
            Student newStudent = new Student("КННІ", "К-01", 3,"Студент");
            Console.WriteLine("Хто ви?");
                //використовуємо властивість
            newStudent.Name = Console.ReadLine();
            Console.WriteLine("Скільки вам років?");
                //використовуємо властивість
            newStudent.Age = int.Parse(Console.ReadLine());
            Console.WriteLine("Прізвище = " + newStudent.name);
            Console.WriteLine("Вік= " + newStudent.age);
            Console.WriteLine("Роль= " + newStudent.Role);
            Console.WriteLine("Факультет = " + newStudent.Faculty);
            Console.WriteLine("група= " + newStudent.Group);
            Console.WriteLine("курс= " + newStudent.Course);
            Console.WriteLine("Ваш рейтинг?");
            string r = Console.ReadLine();
                // привсоєння значення відкритому полю класу
            newStudent.Rating = int.Parse(r);
            newStudent.StudentRating(newStudent.Rating);
            Console.ReadLine();
        }
    }
}
```

2. Програма автоматизованого обліку банківських відомостей

Програма є автоматизованою системою обліку банківських відомостей. На кожного клієнта банку зберігаються наступні відомості:

- прізвище, ім’я, по-батькові;
- дата народження;
- паспортні дані;
- ідентифікаційний код;
- місце роботи (навчання);
- номери рахунків.

Для кожного клієнта визначимо операції:

- додати нового клієнта;
- видалити клієнта;
- змінити реквізити клієнта.

На кожному рахунку зберігається інформація про поточний баланс. З кожним рахунком можна виконувати наступні дії:

- відкриття, закриття;
- внесення грошей, зняття грошей;
- перегляд балансу.

Створимо два класи:

- класс Client для опису інформації про клієнта;
- клас Account для опису банківського рахунку

Клас *Client*. Варіант 1 - конструктор без параметрів, властивості класу

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ex3Lab3
{
    class Client
    {
        //поля класу
        private string name;
        private DateTime birthDate;
        private string passport;
        private string iD;
        private string job;
        private string nomAccount;
        /// <summary>
        /// конструктор без параметрів
        /// </summary>
        public Client()
        {
        }
    }
}
```

Область видимості полів класу відповідно до правил має бути визначена або як **закрита**, або як **захищена**. Доступ же до полів - членів класу має бути організований або за допомогою методів, або за допомогою властивостей класу. Створимо властивості класу *Client*, які забезпечують читання і запис значень закритих полів класу.

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ex3Lab3
{
    class Client
    {
        //поля класу
        private string name;
        private DateTime birthDate;
        private string passport;
        private string iD;
        private string nomAccount;
        /// <summary>
        /// конструктор без параметрів
        /// </summary>
        public Client()
        {
            //поля класу слід ініціалізувати значеннями за замовчуванням
        }
        /// <summary>
        /// властивість класу з методами get, set для закритого поля passport
        /// </summary
        public string Passport
        {
            get
            {return passport; }
            set
            { passport = value; }
        }
        /// <summary>
        /// властивість класу з методами get, set для закритого поля name
        /// </summary>
        public string Name
        {
            get
            { return name; }
            set
            { name = value; }
```

```
        }
        /// <summary>
        /// ////////////////
        /// </summary>
        public int Age
        {
            get
            {
                int yearNumbers; //кількість років
                yearNumbers = DateTime.Now.Year - BirthDate.Year;
                return yearNumbers;
            }
        }
        /// <summary>
        /// методи get і set для закритого поля BirthDate
        /// </summary>
        public DateTime Birthdate
        {
            get
            { return birthDate; }
            set
            {
                if (DateTime.Now > value)
                    birthDate = value;
                else
                    throw new Exception("Введена невірна дата народження");
            }
        }
        /// <summary>
        /// властивість класу з методами get, set для закритого поля iD
        /// </summary>
        public string ID_kod
        {
            get { return iD; }
            set
            { iD = value; }
        }
        /// <summary>
        /// властивість класу з методами get, set для закритого поля nomAccount
        /// </summary>
        public string Nom_Account
        {
            get { return nomAccount; }
            set
            { nomAccount = value; }
        }
    }
}
```

Як видно з прикладу, властивість складається з методів *set* і *get*. При цьому властивість повинна містити хоч би один з методів. Метод *set* дозволяє змінювати значення поля класу, *get* – отримувати значення. У метод *set* передається значення параметра за допомогою змінної *value*. Обидва методи можуть містити довільну кількість операторів, що описують алгоритм виконання дій в процесі читання або запису значення в полі класу. У даному прикладі властивості *Passport* і *Name* дозволяють просто дістати доступ до полів класу, читаючи або встановлюючи значення відповідних змінних. Властивість *Birthdate* також призначена для читання і запису значення змінної - члена класу *BirthDate*. При цьому при читанні значення (операція *get*) відбувається просто передача значення змінної *BirthDate*, при спробі ж запису нового значення в цю змінну відбувається перевірка допустимості встановленого значення змінної. В даному випадку перевірка зводиться до порівняння нового значення дати народження з поточною датою. Якщо встановлене значення дати народження більше або дорівнює поточній даті, генерується виключення, яке не дозволяє записати нове значення в змінну, - член класу.

Властивість *age* застосовується для отримання поточного віку клієнта. Вона призначена лише для читання значення із змінної, тому містить лише метод *get*. При використанні властивості *age* відбувається обчислення поточного значення віку клієнта в роках шляхом віднімання року народження від поточного значення року.

Використання властивостей аналогічно використанню змінних. У наступному прикладі створюється об'єкт *c1* класу *Client*. Потім поля цього об'єкту заповнюються значеннями з використанням властивостей. Після цього на екран виводяться значення полів, для цього також застосовуються властивості класу:

```
using System;
namespace ex3Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            Client c1 = new Client();
            c1.Name = "Вася";
            c1.Passport = "9002";
            c1.Birthdate = new DateTime(2003, 08, 03);
            c1.ID_kod = "123456789";
            c1.Nom_Account = "8097";
            Console.WriteLine("Имя=" + c1.name);
            Console.WriteLine("Паспорт=" + c1.passport);
            Console.WriteLine("Возраст=" + c1.age);
            Console.WriteLine("Иден.код=" + c1.ID_kod);
            Console.WriteLine("Код счета=" + c1.Nom_Account);
            Console.ReadKey();
        }
    }
}
```

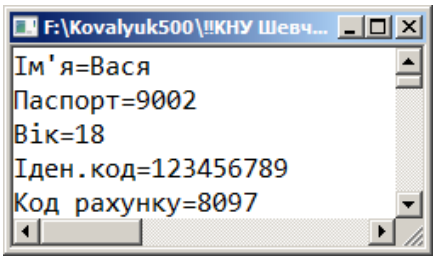

Рисунок 2 – Результат роботи програми 2

Клас Client. Варіант 2 - конструктор класу з параметрами

```
/// <summary>
/// конструктор з параметрами
/// </summary>
/// <param name="clientName">ім'я клієнта</param>
/// <param name="clientPassport">паспорт</param>
/// <param name="clientBirthDate">дата народження</param>
public Client(string clientName, string clientPassport,
                    DateTime clientBirthDate)
{
    name = clientName;
    passport = clientPassport;
    birthdate = clientBirthDate;
}
```

Видно, що конструктор має три параметри. У тілі конструктора відбувається запис переданих як параметри значень у відповідні поля класу за допомогою властивостей даного класу. У випадку з датою народження це дозволяє не дублювати процедуру перевірки введеної дати, а скористатися алгоритмом, реалізованим у властивості *birthdate*.

Створимо також метод, що дозволяє змінити значення полів об'єкту класу *Client*:

```
/// <summary>
/// редагування полів
/// </summary>
/// <param name="clientName">ім'я клієнта</param>
/// <param name="clientPassport">паспорт</param>
/// <param name="clientBirthDate">дата народження</param>
public void EditClient(string clientName, string clientPassport,
                            DateTime clientBirthDate)
{
    name = ClientName;
    passport = ClientPassport;
    birthdate = ClientBirthDate;
}
```

Як видно з прикладу, код цього методу практично повністю ідентичний конструктору з параметрами з різницею в імені, а також в типі значення, яке повертається. Звичайно, в даному випадку

можна було б обійтися і використанням властивостей для зміни значень полів класу, проте, інколи буває корисно, аби такого роду зміни були реалізовані в рамках одного методу, тим більше, якщо алгоритм змін є нестандартним.

Тепер, з використанням конструктора з параметрами, можна створити і відразу ж ініціалізувати об'єкт класу *Client*:

```
Client c1 = new Client("Вася", "9002", new DateTime(2003, 08, 03));
```

Змінений код з викликом редагування полів класу:

```
static void Main(string[] args)
 {
     Client c1 = new Client("Вася", "9002", new DateTime(2003, 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(2000, 09, 01));
     //Виведення значень відредагованих полів
     Console.WriteLine("Виведення значень відредагованих полів");
     Console.WriteLine("Ім'я=" + c1.Name);
     Console.WriteLine("Паспорт=" + c1.Passport);
     Console.WriteLine("Вік=" + c1.Age);
     Console.ReadKey();
 }
```

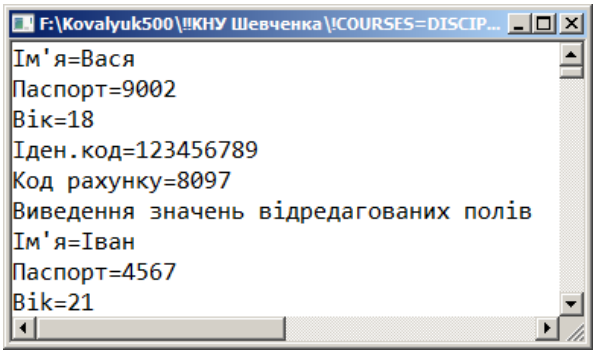

Рисунок 3 – Результат роботи програми 2, варіант 2

Клас Account. Варіант 1

У першому варіанті класу *Account* активно використовуватимемо поля класу. Окрім двох основних полів *credit* і *debit*, які зберігають надходження і витрати рахунку, введемо поле *balance*, яке задає поточний стан рахунку, і два поля, пов'язані з останньою виконуваною операцією. Поле *sum* зберігатиме суму грошей поточної операції, а поле *result* – результат виконання операції. Полів в класу багато, і як наслідок, в методах класу аргументів буде небагато.

```
class Account
{
    //закриті поля класу
    int debit = 0, credit = 0, balance = 0;
    int sum = 0, result = 0;
    /// <summary>
    /// Зарахування на рахунок з перевіркою
    /// </summary>
    /// <param name="sum">сума, що зараховується</param>
    public void PutMoney(int sum)
    {
        this.sum = sum;
        if (sum > 0)
        {
            credit += sum; balance = credit - debit; result = 1;
        }
        else result = -1;
```

```
        Mes();
    }//PutMoney
    /// <summary>
    /// Зняття з рахунку з перевіркою
    /// </summary>
    /// <param name="sum">сума, що знімається</param>
    public void GetMoney(int sum)
    {
      this.sum = sum;
      if (sum <= balance)
      {
         debit += sum; balance = credit - debit; result = 2;
      }
       else result = -2;
        message();
    }//getMoney
   /// <summary>
   /// Повідомлення про виконання операції
   /// </summary>
   void message()
   {
    switch (result)
    {
     case 1:
       Console.WriteLine("Операція зарахування грошей пройшла успішно!");
       Console.WriteLine("Сума={0},Ваш поточний баланс={1}", sum, balance);
       break;
     case 2:
       Console.WriteLine("Операція зняття грошей пройшла успішно!");
       Console.WriteLine("Сума={0},Ваш поточний баланс={1}", sum, balance);
       break;
     case -1:
       Console.WriteLine("Операція зарахування грошей не виконана!");
       Console.WriteLine("Сума має бути більше за нуль!");
       Console.WriteLine("Сума={0},Ваш поточний баланс={1}", sum, balance);
       break;
     case -2:
       Console.WriteLine("Операція зняття грошей не виконана!");
       Console.WriteLine("Сума має бути не більше балансу!");
       Console.WriteLine("Сума={0},Ваш поточний баланс={1}", sum, balance);
       break;
     default:
       Console.WriteLine("Невідома операція!");
       break;
    }
    Console.ReadLine();
  }
}
```

Як можна бачити, лише у методів *getMoney*() і *putMoney*() є один вхідний аргумент. Це той аргумент, який потрібний по суті справи, оскільки лише клієнт може вирішити, яку суму він хоче зняти або покласти на рахунок. Інших аргументів в методів класу немає - вся інформація передається через поля класу. Зменшення числа аргументів призводить до підвищення ефективності роботи з методами, оскільки зникають витрати на передачу фактичних аргументів. Але при цьому ускладнюються операції роботи з вкладом, оскільки потрібно у момент виконання операції оновлювати значення полів класу. Закритий метод *Mes*() викликається після виконання кожної операції, повідомляючи про те, як пройшла операція, і інформуючи клієнта про поточний стан його балансу.

Клас Account. Варіант 2

Спроектуємо аналогічний клас *Account1*, який відрізняється лише тим, що у нього буде менше полів. Замість поля *balance* в класі з'явиться відповідна функція з цим же іменем, замість полів *sum* і *result* з'являться аргументи в методах, що забезпечують необхідну передачу інформації. От як виглядає цей клас:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ex3Lab3
{
 class Account1
 {
   //закриті поля класу
```

```
    int debit = 0, credit = 0;
    /// <summary>
    /// Зарахування на рахунок з перевіркою
    /// </summary>
    /// <param name="sum">зарахована сума</param>
    public void PutMoney(int sum)
    {
        int res = 1;
        if (sum > 0) credit += sum;
        else res = -1;
        message(res, sum);
    } //putMoney
    /// <summary>
    /// Зняття з рахунку  со счета з перевіркою
    /// </summary>
    /// <param name="sum"> сума, що знімається</param>
    public void GetMoney(int sum)
    {
        int res = 2;
        if (sum <= balance()) debit += sum;
        else res = -2;
        balance();
        message(res, sum);
    }//getMoney
    /// <summary>
    /// розрахунок балансу
    /// </summary>
    /// <returns>поточний баланс</returns>
    int balance()
    {
        return (credit - debit);
    }
    /// <summary>
    /// Повідомлення про виконання операції
    /// </summary>
    void message(int result, int sum)
    {
     switch (result)
     {
       case 1:
        Console.WriteLine("Операція зарахування грошей пройшла успішно!");
        Console.WriteLine("Сума={0},Ваш текущий баланс={1}", sum, balance());
        break;
       case 2:
        Console.WriteLine("Операція зняття грошей пройшла успішно!");
        Console.WriteLine("Сума={0},Ваш текущий баланс={1}", sum, balance());
        break;
       case -1:
        Console.WriteLine("Операція зарахування грошей не виконана!");
        Console.WriteLine("Сума має бути більше за нуль!");
        Console.WriteLine("Сума={0},Ваш поточний баланс={1}", sum, balance());
        break;
       case -2:
        Console.WriteLine("Операція зняття грошей не виконана!");
        Console.WriteLine("Сума має бути не більше балансу!");
        Console.WriteLine("Сума={0},Ваш поточний баланс={1}", sum, balance());
        break;
       default:
         Console.WriteLine("Невідома операція!");
        break;
       }
    }
  }//Account1
}
```

Порівнюючи цей клас з класом *Account*, можна бачити, що число полів скоротилося з п'яти до двох, спростилися основні методи *GetMoney*() і P*utMoney*(). Але, як плата, в класі з'явився додатковий метод *balance*(), що багато разів викликається, і в методі *message*() тепер з'явилися два аргументи. Який клас кращий? Однозначно сказати не можна, все залежить від контексту, від пріоритетів, заданих при створенні конкретної системи.

Метод TestAccounts() класу Program*, що тестує роботу з класами* Account *і* Account, має викликатися у методі *Main():*

```
using System;
namespace ex3Lab3
{
    class Program
    {
        static void TestAccounts()
        {
            Account myAccount = new Account();
            myAccount.PutMoney(6000);
            myAccount.GetMoney(2500);
            myAccount.PutMoney(1000);
            myAccount.GetMoney(4000);
            myAccount.GetMoney(1000);
            //Аналогічна робота з класом Account1
            Console.WriteLine("Новий клас і новий рахунок!");
            Account1 myAccount1 = new Account1();
            myAccount1.PutMoney(6000);
            myAccount1.GetMoney(2500);
            myAccount1.PutMoney(1000);
            myAccount1.GetMoney(4000);
            myAccount1.GetMoney(1000);
            Console.WriteLine("Кінець роботи");
            Console.ReadLine();
        }// TestAccounts
        static void Main(string[] args)
        {
            Console.WriteLine("Тестування класів Account і Account1 ");
            TestAccounts();
        }
    }
}
```

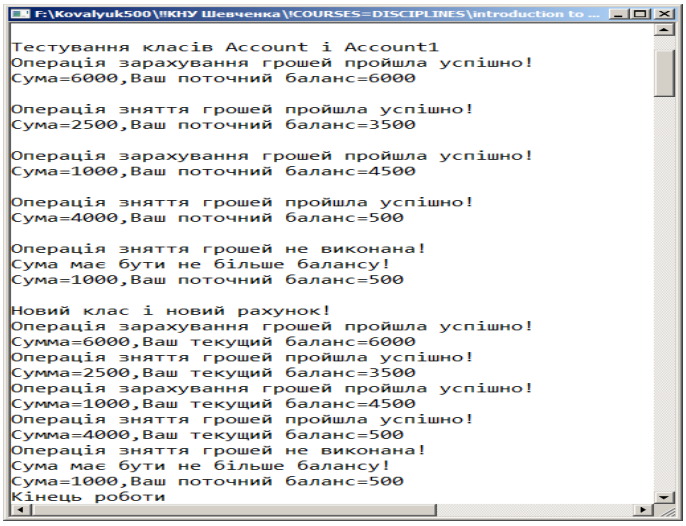

Рисунок 4 – Результат роботи програми. Клас Account. Варіант 2

3. Вбудовані (вкладені) класи

У мові C# будь-який клас у своїй реалізації може містити оголошення іншого класу. Клас, що оголошується в межах фігурних дужок іншого класу, називається *вкладеним класом*. Вкладені класи можуть мати модифікатори доступу *public, protected, internal, protected internal, private* або *private protected*. Об‘єкт вкладеного класу можна оголосити у випадку, якщо вкладений клас оголошений як видимий (не як *private*).

Нехай *Outer* – ім’я класу, який містить в собі оголошення іншого класу з іменем *Inner*, *Inner* – ім’я класу, який оголошується в класі *Outer*. Якщо вкладений клас оголошено як не *private*-клас, то створення екземпляру цього класу має такий вигляд: `Outer.Inner objInner = new Outer.Inner();`

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ex6Lab3
{
    class Outer
    {
        // внутрішні змінні класу Outer
```

```
        public int D;
        static public int Sd;
        // доступ до private-класу Inner1 з внутрішнього методу класу
        public Inner GetInner()
        {
            Console.WriteLine("Метод GetInner() зовнішнього класу");
            Inner i1 = new Inner(); // створити екземпляр класу Inner1
            // доступ до члена даних класу Inner1 через екземпляр класу
            i1.d1 = 25;
            // доступ до статичного члена вкладеного класу Inner1
            Inner.sd1 = 30;
            return i1;
        }
        // private-вкладений клас Inner
        public class Inner
        {
            public int D1;
            public static int Sd1;
            public string Method()
            {
                Console.WriteLine("Метод внутрішнього класу ");
                string str = "  результат вкладеного класу Inner";
                return str;
            }
        }
    }
}
using System;
namespace ex6Lab3
{
  class Program
  {
     static void Main(string[] args)
     {
       Console.WriteLine("Hello World!");
       // Використання вкладених класів Outer.Inner1, Outer.Inner2, Outer
       // 1. Оголосити об'єкт класу Outer
       Outer o = new Outer();
       o.D = 300; // доступ до змінної через екземпляр класу Outer
       Outer.Sd = 230; // доступ до статичної змінної
       Console.WriteLine("Поля зовнішнього класу: "+ o.D+" "+ Outer.Sd);
       Console.WriteLine("Main:"+ o.GetInner());
       // 2. Оголосити об'єкт public-класу Inner
       Outer.Inner i1 = new Outer.Inner();
       i1.D1 = 440;
       Outer.Inner.Sd1 = 500; // доступ до статичної змінної внутрішнього класу
       Console.WriteLine("Поля внутрішнього класу " + i1.D1 + " " +
    Outer.Inner.Sd1);
       Console.WriteLine("Main:"+i1.Method());
      }
  }
}
```

4. Часткові класи (класи, що розділяються) і часткові методи

Класи можуть бути частковими. Тобто ми можемо мати кілька файлів з визначенням одного і того самого класу, і при компіляції всі ці визначення будуть скомпільовані в одне. Наприклад, визначимо в проекті два файли з кодом.

```
// частина 1 класу Person
public partial class Person
{
    public void Move()
    {
        Console.WriteLine("I am moving");
    }
}
// частина 2 класу Person
public partial class Person
{
    public void Eat()
    {
        Console.WriteLine("I am eating");
```

```
    }
}
```

Отже, два файли в проекті містить визначення одного і того самого класу *Person*, які містять два різних методу. І обидва визначені класи є **частковими**. Для цього вони визначаються з ключовим словом ***partial***.

Ключове слово *partial* вказує, що інші частини класу, структури або інтерфейсу можуть бути визначені в просторі імен. Всі частини повинні використовувати ключове слово *partial*. Для формування остаточного класу всі частини повинні бути доступні під час компіляції. Всі частини повинні мати однакові модифікатори доступу, наприклад *public*, *private* тощо.

Приклад використання методів часткових класів:

```
class Program
{
    static void Main(string[] args)
    {
        Person Vadim = new Person();
        Vadim.Move(); // метод першого часткового класу
        Vadim.Eat();  // метод другого часткового класу
        Console.ReadKey();
    }
}
```

Часткові класи можуть містити часткові методи. Такі методи також визначаються з ключовим словом *partial*. Причому визначення часткового методу без тіла методу знаходиться в одному частковому класі, а реалізація цього самого методу – в іншому частковому класі..

```
public partial class Person
{
    partial void DoSomething(); //оголошення часткового методу
    public void OneMethod()
    {
        Console.WriteLine("Start");
        DoSomething(); //call method of second partion class
        Console.WriteLine("Finish");
    }
}
public partial class Person
{
    //визначення часткового методу
    partial void DoSomething()
    {
        Console.WriteLine("I am reading a book");
    }
}
```

Як правило, часткові методи завжди закриті (private). Виклик часткових методів здійснюється через відкриті методи

```
Person Ivan = new Person();
Ivan. OneMethod(); //метод викликає закритий частковий метод
```

5. Текстові файли

В C# для роботи з файлами і потоками використовуються класи ***FileStream, StreamWriter***, ***StreamReader, FileInfo*** і інші класи.

Для введення і виведення даних не потоками, а рядками призначені класи *StringReader*, *StringWriter*.

Розглянемо приклад**.** В цьому прикладі спочатку створюється об'єкт ***f*** класу *StreamWriter*, який містить файл для виводу **output.txt**. Якщо не вказати повний шлях до файлу як нашому прикладі, то файл буде створений в директорії, де розміщується exe-файл програми: *...\\bin\\debug\\output.txt*

Далі визначаються і ініціалізуються змінні ***int i*** та ***string s***, які записуються у вихідний файл. Метод *Close()* закриває файл. В другій частині прикладу по черзі зчитуються рядки з файлу *input.txt*, перетворюються у числові типи і виводяться на консоль.

```
using System;
using System.IO; // підключення простору імен для роботи з файлами
namespace ex7Lab3
{
 class Program
```

```
{
    static void Main(string[] args)
    {
        //запис у текстовий файл
        Console.WriteLine("Hello World!");
        StreamWriter fout = new StreamWriter("output.txt");      // 2
                                              //відкрити файл для запису
        int i = 3;
        string s1 = "Мова програмування C# - це C++++ ";
        fout.WriteLine("i = " + i);                                 // 3
        fout.WriteLine("s1 = " + s1);                                 // 4
        fout.Close();                                               // 5
          //=========читання з текстового файлу================
        StreamReader fin = new StreamReader("f:\\input.txt"); //  відкрити
                                  //файл для читання
        string s2 = fin.ReadLine(); // читати рядок
        Console.WriteLine("s = " + s2);
        char c = (char)fin.Read(); // читати символ
        fin.ReadLine();
        Console.WriteLine("c = " + c);
        string buf = fin.ReadLine(); //читати рядок і конвертувати у ціле число
        int j = Convert.ToInt32(buf);
        Console.WriteLine(j);
        buf = fin.ReadLine(); //читати рядок і конвертувати у число double
        double x = Convert.ToDouble(buf);
        Console.WriteLine(x);
        buf = fin.ReadLine(); //читати рядок і конвертувати у число double
        double y = double.Parse(buf);
        Console.WriteLine(y);
        buf = fin.ReadLine(); //читати рядок і конвертувати у ціле число
        decimal z = decimal.Parse(buf);
        Console.WriteLine(z);
        fin.Close();
       }
   }
}
```

Варіанти завдань для лабораторної роботи № 3

Інструкція з виконання лабораторного завдання

Виконання лабораторної роботи передбачає три етапи (рис.4).

Етап 1 – об’єктно-орієнтований аналіз (ООА) з виконанням об’єктно-орієнтованої декомпозиції предметної області. Об'єктно-орієнтований аналіз дозволяє чітко визначити основні сутності системи та їхню поведінку. На вході - опис предметного середовища Результат – модель предметної області у вигляді сукупності сутностей з атрибутами та операціями, що характеризують поведінку сутностей, відповідно до принципів SOLID.

Етап 2 – об’єктно-орієнтоване проєктування (OOD). На входе результат об’єктно-орієнтованої декомпозиції. Результат – UML діаграма класів, яка містить класи і взаємозв'язки між ними та UML Use Case діаграма для визначення сценарію роботи програми.

Етап 3 – об’єктно-орієнтоване програмування (ООР). На вході UML діаграма класів і UML Use Case діаграма. Результат – код програми відповідно до вимог чистого коду (гарного стилю програмування) та принципів SOLID (в цій лабораторній роботі слід дотримуватись принципів єдиної відповідальності (SRP), та принципу відкритості – закритості (OCP)).Виконання лабораторної роботи передбачає три етапи.

Етап 1 – об’єктно-орієнтований аналіз (ООА).

На вході етапу ООА розробник має опис предметної області, в якій описана бізнес-логіки системи. На виході як результат ООА – модель предметної області відповідно до принципів SOLID з визначенням класів, їх атрибутів, об’єктів та методів, що визначають їх поведінку.

Етап 2 – об’єктно-орієнтоване проєктування (OOD).

На вході етапу OOD модель предметної області як результат етапу ООА. На виході етапу OOD – *UML діаграма класів* з переліком усіх класів, атрибутів, операцій (методів) та взаємозв'язків між класами. Для лабораторної роботи 3 рекомендується встановлювати **асоціативні** (направлені і ненаправлені) зв’язки між класами. Кратність зв’язків **один до одного**. **Не використовуйте зв’язки типу успадкування, агрегація, композиція.**

Етап 3 – об’єктно-орієнтоване програмування (ООР).

На вході етапу ООР маємо UML діаграму класів (результат етапу OOD). На виході етапу ООР– код класів відповідно до вимог чистого коду (гарного стилю програмування) та принципів SOLID (в цій лабораторній роботі слід дотримуватись принципів *єдиної відповідальності (SRP), та принципу відкритості – закритості (OCP)*). Сценарій роботи програми реалізується у функції Main() або розробник реалізує його в окремому класі. Для розробки сценарію роботи програми необхідно розробити UML діаграму варіантів використання (Use Case Diagram)

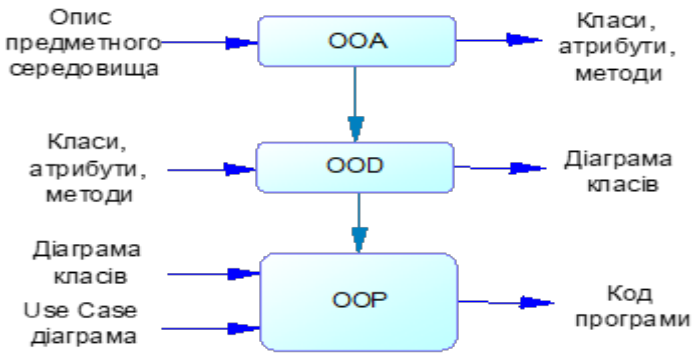

Рисунок 4 – Етапи об'єктно-орієнтованого підходу до розробки ПЗ

Для забезпечення SOLID принципу відкритості – закритості (OCP) потрібно створити 4 версії коду лабораторної роботи №3. Кожна версія коду в окремому проєкті:

- версія 1 включає пункти 1,2,3,4,5,6. Дедлайн зараховується по версії 1.
- версія 2 включає код версії 1 та пункт 7 завдання;
- версія 3 включає код версії 2 та пункт 8 завдання;
- версія 4 включає код версії 3 та пункт 9 завдання.

Для забезпечення SOLID принципу єдиної відповідальності (SRP) слід визначити методи, які властиві зазначеним об’єктам предметної області. Методи, які не властиві відповідним об’єктам, слід перенести в інші класи, визначивши їх самостійно. Для опрацювання полів класів, які не задіяні в заданих предметною областю методах класів, їх слід або вилучити, або задати методи, які ці поля використовуватимуть самостійно. Якщо студент вважає для зручності роботи користувача з програмою

використовувати меню, для цього слід розробити окремий клас Menu. Не ускладнюйте інші класи, включаючи меню у вигляді метода в клас.

На етапі **об'єктно-орієнтоване програмування (ООР)** потрібно розробити код відповідно до пунктів завдання.

Пункт 1 завдання передбачає розробку класів відповідно до опису предметної області, що містять закриті поля.

Пункти 2 та 3 завдання передбачають розробку конструкторів за замовчування, з параметрами (та конструктора копії) в створених класах.

Пункт 5 завдання – розробка в створених класах відкритих методів за сценарієм роботи програми (UML Use Case diagram). Алгоритми методів студент розробляє самостійно. В завданні алгоритм може бути не описаний.

Пункт 6 завдання – створення сервісного класу, в який включити відкриті методи для читання з консолі, виведення результатів на консоль, методів запису та читання з текстових файлів.

Пункт 7 завдання – доповнення класів вбудованими (вкладеними класами). Рекомендується у вигляді другої версії лабораторної роботи.

Пункт 8 завдання – доповнення другої версії програми частковими класами та методами. Зробити у вигляді третьої версії лабораторної роботи.

Пункт 9 завдання – доповнення третьої версії новим статичним класом.

В методі *Main*() класу *Program* продемонструвати виклики усіх методів усіх класів. Усі значення, що розраховуються, записувати до *текстових файлів,* методи їх обробки включити в сервісний клас, оскільки ці методи не властиві об'єктам предметної області.

Студент має право додати додаткові поля та методи в класи завдань свого варіанту. Номер варіанта визначається за порядковим номером студента в журналі групи.

<table>
<tr><th>№ варіанту</th><th>Зміст завдання</th></tr>
<tr><td>1.</td><td>
Опис предметної області. Існує освітнє середовище, в якому учасниками освітнього процесу є викладач і студент. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних учасникам освітнього середовища операцій. В таблиці 1.1 подані ролі, атрибути та операції сутностей освітньогго процесу.
Таблиця 1.1. Основні сутності предметної області
<table>
<tr><th>Сутність</th><th>Роль сутності</th><th>Атрибути сутності</th><th>Операції (дії) сутності</th></tr>
<tr><td>Викладач</td><td>Джерело навчального контенту; координує дисципліну; виконує оцінювання студентів</td><td>Ім'я викладача
Назва дисципліни
Навчальне навантаження (години)
Кількість студентів</td><td>Збільшити кількість студентів
Зменшити кількість студентів
Змінити обсяг навчального навантаження
Поставити оцінку студенту
Передати матеріал
Записати оцінку в журнал</td></tr>
<tr><td>Студент</td><td>Отримувач знань і навчального контенту; виконує завдання; формує власний рейтинг</td><td>Ім'я студента
Назва дисципліни
Список робіт з оцінками
Обсяг виконаних робіт</td><td>Додати оцінку
Переглянути оцінки
Розрахувати рейтинг
Завантажити матеріал</td></tr>
<tr><td>Сервіс</td><td>Технічний посередник: забезпечує обмін даними, введення/виведення, зберігає дані</td><td>Формат виводу
Шлях до файлу
Дані для обробки</td><td>Вивести дані на консоль
Читати дані з консолі
Записати дані у файл
Читати дані з файлу</td></tr>
</table>
Здійснити об'єктно-орієнтований аналіз і визначити класи, поля і методи. Спрощена взаємодія між учасниками освітнього середовища: один викладач працює з одним студентом. Сервіс не є учасником освітнього процесу, є лише інструментом, який використовують інші сутності (викладач і студент) для реалізації своїх дій.
Розробити таблицю зв'язків між сутностями за зразком:
<table>
<tr><td>Сутність 1</td><td>Сутність 2</td><td>Тип зв'язку</td><td>Опис зв'язку між сутностями</td></tr>
</table>
На основі результатів ООА здійснити об'єктно-орієнтоване проєктування і побудувати діаграму класів. На основі діаграми класів розробити код класів мовою C# для обробки даних про викладача і студента. Розробити сценарій роботи програми і побудувати Use Case діаграму.
На основі Use Case діаграми написати код для реалізації сценарію роботи програми у функції Main() програмного моделювання предметної області «Освітній процес 1».
Версія 1 (пункти завдання 1,2,3,4,5,6).
</td></tr>
</table>

1. Створити класи **Teacher, Student, Service**. Кожен клас має бути створений в окремому файлі командою *Project → Add class.* Визначити в класах закриті поля, що відповідають атрибутам предметної област (табл.1.1).
2 Визначити в класах конструктор за замовчуванням (без параметрів для ініціалізації полів класів нульовими та пустими (для типу *string*) значеннями).
3. Визначити в класах конструктори з параметрами для ініціалізації полів класів початковими значеннями та конструктор копії для створення нових об'єктів як копії існуючого об'єкта.
4. Визначити в класах відкриті властивості (get, set) для доступу до закритих полів та зміни значень ініціалізованих в конструкторах полів класів.
5. Визначити в класах відкриті методи для реалізації поведінки об'єктів через зміну станів об’єктів відповідно до табл.1.1.
6. У функції Main() реалізувати сценарій роботи програми відповідно до Use Case діаграми.

Версія 2 (Версія 1 + п.7) в окремому проєкті

7. Додати до класу **Student** вбудований (вкладений) клас Дипломний_проект (**DiplomaProject**) з додатковими сутностями згідно з табл. 1.3:

Таблиця 1.3. Додаткові сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності
Дипломний проект	Інтегрує знання студента й оцінюється викладачем	Назва теми, Кількість реалізованих алгоритмів, Складність теми. Оцінка Ім’я керівника	Вибрати тему з переліку. Визначити складність теми. Визначити оцінку

Алгоритм вибору теми такий: перелік тем подати текстовим файлом, з консолі ввести ключові слова, читати рядки файлу в пам’ять, шукати входження ключових слів в прочитані рядки, вивести на консоль рядок, в якому знайдені задані ключові слова.

Алгоритм визначення оцінки дипломного проекту (ДП): порахувати кількість реалізованих в ДП методів, задати складність кожного методу, визначити складність усього ДП, визначити співвідношення складності ДП та балів, порахувати оцінку в балах.

Версія 3 (Версія 2 + п.8) в окремому проєкті.

8. Модифікувати клас **Student**, подавши його як частковий, поділивши клас на дві частини кожна в окремому файлі: в один файл включити метод вибору теми, в інший файл – метод розрахунку оцінки за дипломний проект.

Версія 4 (Версія 3 + п.9) в окремому проєкті.

9. Додати до проекту новий статичний клас Наукова стаття (**ScientificPaper**), включивши в нього функцію (на вибір студента) з варіанта 1 лабораторної роботи 2, зробивши її статичною.

Алгоритми методів, які не описані в завданні, студент розробляє самостійно.

2.

Опис предметної області. Існує освітнє середовище, в якому учасниками освітнього процесу є кафедра і студент. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних участникам освітнього середовища операцій. В таблиці 2.1 подані ролі, атрибути та операції сутностей освітньогго процесу.

Таблиця 2.1. Основні сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умова виконання дії
Кафедра	Організаційна одиниця, яка адмініструє напрям підготовки та облік студентів	Назва кафедри Кількість студентів Напрям підготовки Перелік дисциплін Максимальна кількість студентів	Збільшити кількість студентів в межах максимальної кількості та з відповідним напрямом підготовки Зменшити кількість студентів Додати дисципліну, що відповідають напрямку Видалити дисципліну	Дія дозволена, якщо поточна кількість < максимальної кількості Додати дисципліну можна, якщо вона відповідає напрямку підготовки кафедри Видалення можливе, якщо дисципліна присутня у списку
Студент	Учасник освітнього процесу, чия успішність вимірюється на основі оцінок	Ім’я студента Напрям підготовки Список оцінок Рівень навчального навантаження Номер залікової книжки	Визначити рейтинг успішності Переглянути оцінки Додати оцінку	Додати оцінку можна, якщо вона не перевищує рейтинг в 100 балів
Сервіс	Технічний посередник: забезпечує обмін, введення /виведення та збереження даних	Формат виводу Шлях до файлу Дані для обробки	Вивести дані на консоль Читати дані з консолі Записати дані у файл Читати дані з файлу	

Здійснити **об'єктно-орієнтований аналіз** і визначити класи, поля і методи. Спрощена взаємодія між учасниками освітнього середовища: **одна кафедра навчає одного студента. Сервіс** не є учасником освітнього процесу, є лише інструментом, який використовують інші сутності (кафедра і студент) для реалізації своїх дій.
Розробити таблицю зв'язків між сутностями за зразком:

Сутність 1	Сутність 2	Тип зв'язку	Опис зв'язку між сутностями

На основі результатів ООА здійснити **об'єктно-орієнтоване проєктування** і побудувати діаграму класів.
На основі діаграми класів розробити **код класів** мовою C# для обробки даних про кафедру і студента.
Розробити **сценарій роботи програми** і побудувати Use Case діаграму.
На основі Use Case діаграми написати код для реалізації сценарію роботи програми у функції Main() програмного моделювання предметної області «Освітній процес 2».

Версія 1 (пункти завдання 1,2,3,4,5,6).

1. Створити класи **Department, Student, Service**. Кожен клас має бути створений в окремому файлі командою *Project → Add class.* Визначити в класах закриті поля, що відповідають атрибутам предметної област (табл.2.1).
2. Визначити в класах конструктор за замовчуванням (без параметрів для ініціалізації полів класів нульовими та пустими (для типу *string*) значеннями).
3. Визначити в класах конструктори з параметрами для ініціалізації полів класів початковими значеннями та конструктор копії для створення нових об'єктів як копії існуючого об'єкта.
4. Визначити в класах відкриті властивості (get, set) для доступу до закритих полів та зміни значень ініціалізованих в конструкторах полів класів.
5. Визначити в класах відкриті методи для реалізації поведінки об'єктів через зміну їх станів відповідно до табл.2.1.

Алгоритм методів збільшення та зменшення кількості студентів, які навчаються на кафедрі такий: згенерувати оцінку з акредитації кафедри (A або B, або E, або F): оцінка A – збільшення кількості студентів на 20%, оцінка B – кількість студентів не змінюється, оцінка E - кількість студентів зменшується на 10%, оцінка F - повідомлення про ліквідацію ліцензії та зменшення кількості студентів на 50%.

6. У функції Main() реалізувати сценарій роботи програми відповідно до Use Case діаграми.

Версія 2 (Версія 1 + п.7) в окремому проєкті

7. Додати до класу **Student** вбудований (вкладений) клас КонкурснаРобота (**CompetitiveWork**) з додатковими сутностями згідно з табл. 2.2:

Таблиця 2.2. Додаткові сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності
Конкурсна робота студента	Творчий результат студента, який оцінюється в межах конкурсу і відображає індивідуальну активність	Назва роботи Тематика конкурсу Дата подання Оцінка / рейтинг Статус (подано, прийнято, відхилено) Призове місце Автор (студент)	Подати роботу Визначити відповідність теми роботи тематиці конкурсу Оцінити роботу Призначити статус Присвоїти місце Переглянути деталі

Алгоритм «Подати роботи». Студент вводить назву роботи, спеціальність. Фіксується поточна дата як дата подання. Статус роботи встановлюється як "подано". Робота зв'язується з автором (студентом). Перевіряється, чи студент не подав іншу роботу (у моделі 1:1). Якщо умови виконані — запис зберігається.
Алгоритм перевірки відповідності теми роботи тематиці конкурсу. Перелік тематик конкурсу подати рядками в текстовому файлі, читаючи рядки з файлу, шукати збіги ключових слів з назви конкурсної роботи з прочитаним рядком з файлу, вивести на консоль рядки файлу, в яких знайдені збіги ключових слів з назви роботи з тематикою конкурсу.
Алгоритми інших дій студент розробляє самостійно.

Версія 3 (Версія 2 + п.8) в окремому проєкті.

8. Модифікувати клас **Student**, подавши його як частковий, поділивши клас на дві частини кожна в окремому файлі: в один файл включити метод перевірки відповідності назви конкурсної роботи тематиці конкурсу, в інший файл – метод розрахунку балів за конкурсну роботу.

Версія 4 (Версія 3 + п.9) в окремому проєкті.

9. Додати до проекту новий статичний клас **CreativeWork**, включивши в нього функцію (на вибір студента) з варіанта 2 лабораторної роботи 2, зробивши її статичною.

Алгоритми методів, які не описані в завданні, студент розробляє самостійно.

3.

Опис предметної області. Існує освітнє середовище, в якому учасниками освітнього процесу є кафедра і викладач. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних участникам освітнього середовища операцій. В таблиці 3.1 подані ролі, атрибути та операції сутностей освітньогго процесу.

Таблиця 3.1. Основні сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умова виконання дії
Кафедра	Керує ресурсами: планує навантаження, розподіляє дисципліни, забезпечує відповідність нормам	Назва кафедри Загальне річне навантаження (год) кафедри Поточна кількість викладачів Перелік дисциплін Мінімальний обсяг дисципліни (90 год) Максимальне навантаження на одного викладача (600 год)	Додати викладача (з перевіркою ліміту) Видалити викладача Розподілити навантаження між викладачами Переглянути розподіл навантаження	Додати викладача можливо, тільки якщо на нього припадає не більше 600 год. Додати викладача можливо, якщо його спеціалізація відповідає профілю дисципліни. Кафедра не може створювати дисципліни з меншим навантаженням ніж 90 годин
Викладач	Реалізує навчальну, наукову та методичну діяльність в межах нормативного навантаження	ПІБ Спеціалізація Загальне річне навантаження (год) викладача Кількість дисциплін Кількість наукових статей Кількість методичних робіт Кількість організаційних робіт	Визначити рейтинг викладача Перевірити перевищення навантаження Додати наукову/методичну роботу	рейтинг викладача = статті × 10 + методичні роботи × 5 + організаційні роботи × 3
Сервіс	Технічний посередник: забезпечує обмін даними, введення/виведення, зберігає дані	Формат виводу Шлях до файлу Дані для обробки	Вивести дані на консоль Читати дані з консолі Записати дані у файл Читати дані з файлу	

Здійснити **об’єктно-орієнтований аналіз** і визначити класи, поля і методи. Спрощена взаємодія між учасниками освітнього середовища: **один викладач працює на одній кафедрі** (модель 1:1). **Сервіс** не є учасником освітнього процесу, є лише інструментом, який використовують інші сутності (кафедра і викладач) для реалізації своїх дій.

Розробити таблицю зв’язків між сутностями за зразком:

Сутність 1	Сутність 2	Тип зв’язку (1:1)	Опис зв’язку між сутностями

На основі результатів ООА здійснити **об’єктно-орієнтоване проєктування** і побудувати діаграму класів. На основі діаграми класів розробити **код класів** мовою C# для обробки даних про кафедру і студента. Розробити **сценарій роботи програми** і побудувати Use Case діаграму.

На основі Use Case діаграми написати код для реалізації сценарію роботи програми у функції Main() програмного моделювання предметної області «Освітній процес 3».

Версія 1 (пункти завдання 1,2,3,4,5,6).

1. Створити класи **Department, Teacher, Service**. Кожен клас має бути створений в окремому файлі командою *Project → Add class.* Визначити в класах закриті поля, що відповідають атрибутам предметної област (табл.3.1).
2. Визначити в класах конструктор за замовчуванням (без параметрів для ініціалізації полів класів нульовими та пустими (для типу *string*) значеннями).
3. Визначити в класах конструктори з параметрами для ініціалізації полів класів початковими значеннями та конструктор копії для створення нових об'єктів як копії існуючого об'єкта.
4. Визначити в класах відкриті властивості (get, set) для доступу до закритих полів та зміни значень ініціалізованих в конструкторах полів класів.
5. Визначити в класах відкриті методи для реалізації поведінки об'єктів через зміну їх станів відповідно до табл.3.1.

Алгоритм методу розподілу навантаження. Згенерувати число, що означає загальне навантаження кафедри в діапазоні від 6000 до 10000 годин. Перевірити можливості розподілу. Обчислити базовий розподіл як загальне навантаження кафедри / кількість викладачів і порівняння його з мінімальним обсягом дисципліни. Перевірити, що не порушені норми (навантаження викладача <= 600), не

порушений мінімальний обсяг дисципліни (обсяг >= 90), загальне навантаження кафедри <= кількість викладачі*600.

6. У функції Main() реалізувати сценарій роботи програми відповідно до Use Case діаграми.

Версія 2 (Версія 1 + п.7) в окремому проєкті

7. Додати до класу **Teacher** вбудований (вкладений) клас Керівництво конкурсними роботами (**ContestWorkManage**r) з додатковою сутністю згідно з табл. 3.2:

Таблиця 3.2. Додаткові сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності
Конкурсна робота студента	Відображає процес супроводу та оцінки конкурсної роботи викладачем	Назва конкурсної Тема конкурсу Статус (прийнята / відхилена) Рейтинг роботи Призове місце, отримане конкурсною роботою (1,2, 3). Значення 0 - робота не попала в номінацію призових. Викладач-керівник	Подати роботу Визначити відповідність теми роботи тематиці конкурсу Оцінити роботу Призначити статус Присвоїти місце Переглянути деталі

Приклад деяких алгоритмів. **Алгоритм «Подати роботи»**. Викладач вводить назву роботи, спеціальність. Фіксується статус роботи встановлюється як «подано».

Алгоритм перевірки відповідності теми роботи тематиці конкурсу. Перелік тематик конкурсу подати рядками в текстовому файлі, читаючи рядки з файлу, шукати збіги ключових слів з назви конкурсної роботи з прочитаним рядком з файлу, вивести на консоль рядки файлу, в яких знайдені збіги ключових слів з назви роботи з тематикою конкурсу.

Алгоритм розрахунку рейтингу конкурсної роботи: згенерувати 3 цілих числа в діапазоні від 0 до 10 (числа свідчать про наукову новизну, практичну цінність, якість оформлення), Сума балів від 25 до 30 свідчить про перше призове місце, сума балів від 20 до 24 – друге призове місце, сума балів від 15 до 19 – третє призове місце, сума балів менше за 15 – робота в номінацію не попала .

Алгоритми інших дій студент розробляє самостійно.

Версія 3 (Версія 2 + п.8) в окремому проєкті.

8. Модифікувати клас **Teacher**, подавши його як частковий, поділивши клас на дві частини кожна в окремому файлі: в один файл включити метод перевірки відповідності назви конкурсної роботи тематиці конкурсу, в інший файл – метод розрахунку рейтингу конкурсної роботи.

Версія 4 (Версія 3 + п.9) в окремому проєкті.

9. Додати до проекту новий статичний клас Міжнародний проект **InterProject**, включивши в нього функцію (на вибір студента) з варіанта 3 лабораторної роботи 2, зробивши її статичною.

Алгоритми методів, які не описані в завданні, студент розробляє самостійно.

4.

Опис предметної області. Існує модель державно-приватного партнерства, в якій учасниками моделі є ІТ-компанія і кафедра. ІТ-компанія створює кафедру. Кафедра укладає партнерство з факультетом. Факультет делегує навчальне навантаження кафедрі. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних учасникам освітнього середовища операцій. В таблиці 4.1 подані ролі, атрибути та операції сутностей моделі державно-приватного партнерства

Таблиця 4.1. Основні сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умова виконання дії
ІТ-компанія	Комерційна організація, яка створює або інтегрує освітні підрозділи для підготовки фахівців	Назва компанії Сфера діяльності Закріплена кафедра Партнерський факультет Якість практичної підготовки студентів	Створити кафедру Закріпити її за напрямом Погодити співпрацю з факультетом Покращення якості практичної підготовки студентів	Наявність договору / меморандуму про співпрацю із ЗВО. Затвердження програм практик. Визнання компанії як бази практики / стажування.

Факультет	Академічна одиниця ЗВО, яка співпрацює з ІТ-компанією для передачі частини підготовки фахівців	Назва факультету Декан Кількість кафедр Загальне річне навчальне навантаження Перелік напрямів Партнерська ІТ-компанія Якість освітнього процесу	Затвердити список напрямів підготовки Визначити обсяг годин на рік, Розподілити річне навантаження між кафедрами Погодити співпрацю з ІТ-компанією Контролювати якість освітнього процесу	
Сервіс	Технічний посередник: забезпечує обмін даними, введення/виведення, зберігає дані	Формат виводу Шлях до файлу Дані для обробки	Вивести дані на консоль Читати дані з консолі Записати дані у файл Читати дані з файлу	У файл потрібно записати значення атрибутів сутностей, все, що буде виведено на консоль, навчальний план, напрям підготовки тощо

Здійснити **об'єктно-орієнтований аналіз** і визначити класи, поля і методи. Спрощена взаємодія між учасниками освітнього середовища: **один факультет співпрацює з однією ІТ-компанією** (модель 1:1). **Сервіс** не є учасником освітнього процесу, є лише інструментом, який використовують інші сутності (кафедра і викладач) для реалізації своїх дій.

Розробити таблицю зв'язків між сутностями за зразком:

Сутність 1	Сутність 2	Тип зв'язку (1:1)	Опис зв'язку між сутностями

На основі результатів ООА здійснити **об'єктно-орієнтоване проєктування** і побудувати діаграму класів.

На основі діаграми класів розробити **код класів** мовою C# для обробки даних про кафедру і студента.

Розробити **сценарій роботи програми** і побудувати Use Case діаграму.

На основі Use Case діаграми написати код для реалізації сценарію роботи програми у функції Main() програмного моделювання предметної області «Освітній процес 3».

Версія 1 (пункти завдання 1,2,3,4,5,6).

1. Створити класи **Faculty, ITCompany, Service**. Кожен клас має бути створений в окремому файлі командою *Project → Add class.* Визначити в класах закриті поля, що відповідають атрибутам предметної област (табл.4.1).
2. Визначити в класах конструктор за замовчуванням (без параметрів для ініціалізації полів класів нульовими та пустими (для типу *string*) значеннями).
3. Визначити в класах конструктори з параметрами для ініціалізації полів класів початковими значеннями та конструктор копії для створення нових об'єктів як копії існуючого об'єкта.
4. Визначити в класах відкриті властивості (get, set) для доступу до закритих полів та зміни значень ініціалізованих в конструкторах полів класів.
5. Визначити в класах відкриті методи для реалізації поведінки об'єктів через зміну їх станів відповідно до табл.4.1.

Алгоритм «Погодити співпрацю з ІТ-компанією». Компанія ініціює запит (формат взаємодії, перелік напрямів, вигоди) про співпрацю,звертаючись до факультету. факультет реєструє звернення, оцінює пропозиції (перевіряє відповідність тематики компанії напряму підготовки факультету, практичну цінність, ресурси). Факультет підписує угоду/меморандум/наказ, якщо схвалена пропозиція, повідомлення про відмову, якщо пропозиція не схвалена.

Алгоритм «Контролювати якість освітнього процесу». Визначити перелік критеріїв якості (дисципліни, кваліфікація викладачі, метод. забезпечення, бажання студентів). Порівняти дані за критеріями з нормативами (можна задати випадковим чином). Формувати висновки, визначивши значення «відповідає критеріям», «має суттєві недоліки», «має несуттєві недоліки». Рішення записується у файл.

Алгоритм «Покращити якість підготовки студентів». Показник якості підготовки студентів (індекс KPI) розраховувати як *Індекс KPI = ((Факт – База) / (Норма – База)) * 100%,* де *Факт* – фактичні результати роботи; *База* – допустиме мінімальне значення показника. Нижче базового рівня – відсутність результату; *Норма* – плановий рівень, те, що студент повинен виконувати обов'язково, нижче – студент не впорався зі своїми обов'язками; *Ціль* – значення, до якого потрібно прагнути, наднормативний показник, що дозволяє поліпшити результати.

6. У функції Main() реалізувати сценарій роботи програми відповідно до Use Case діаграми.

Версія 2 (Версія 1 + п.7) в окремому проєкті

6. Додати до класу **Faculty** вбудований (вкладений) клас Кафедра (**Department**) з додатковою сутністю згідно з табл. 4.2:

Таблиця 4.2. Додаткові сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умови виконання дій
Кафедра	Внутрішній підрозділ факультету, що реалізує навчальне навантаження у співпраці з факультетом	Назва кафедри Спеціалізація Завідувач кафедри Кількість викладачів Перелік дисциплін з обсягом годин Річне навчальне навантаження Партнерський факультет	Сформувати навчальний план (перелік дисциплін та їх години) Призначити викладачів Погодити напрям навчання	Призначити викладача на дисципліну можливо, якщо його спеціалізація відповідає профілю дисципліни. Напрям навчання погоджений, якщо є відповідні кадри

Алгоритм «Сформувати навчальний план». Отримати від факультету загальне навчальне навантаження. Скласти список дисциплін з обсягом годин. Визначити послідовність їх викладання та розбити дисципліни по роках та семестрах. Критерії: в семестрі 30 кредитів (900 годин). В року 60 кредитів (1800 годин). Затвердити на факультеті.

Алгоритми інших дій студент розробляє самостійно.

Версія 3 (Версія 2 + п.8) в окремому проєкті.

8. Модифікувати клас **Faculty**, подавши його як частковий, поділивши клас на дві частини кожна в окремому файлі: в один файл включити метод формування навчального плану, в інший файл – метод призначення викладачів на дисципліну

Версія 4 (Версія 3 + п.9) в окремому проєкті.

9. Додати до проекту новий статичний клас **CompanyProject** (проект IT-компанії), включивши в нього функцію (на вибір студента) з варіанта 4 лабораторної роботи 2, зробивши її статичною.

Алгоритми методів, які не описані в завданні, студент розробляє самостійно.

5.

Опис предметної області. Існує інституційно-адміністративна модель, в якій сутностями моделі є університет і факультет. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних учасникам освітнього середовища операцій. В таблиці 5.1 подані ролі, атрибути та операції сутностей інституційно-адміністративна моделі.

Таблиця 5.1. Основні сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умова виконання дії
Університет	Головна академічна установа, що координує факультети, формує рейтинг і бюджет	Назва університету Річний бюджет Рейтинг (за Osvita.ua) Кількість факультетів Кількість студентів, наукових праць, міжнародних проєктів, ЗНО вступників	Розрахувати рейтинг університету за версією Osvita.ua Розрахувати розмір річного фінансування за рейтингом Додати факультет Вилучити факультет	Рейтинг оновлюється один раз на рік. рейтинг = f(середній бал ЗНО, кількість публікацій, кількість студентів, кількість факультетів * середня ефективність) Фінансування залежить від позиції у рейтингу: чим вище — тим більша частка бюджету.
Факультет	Академічна одиниця ЗВО, яка співпрацює з IT-компанією для передачі частини підготовки фахівців	Назва факультету Декан Кількість кафедр Кількість спеціальностей Кількість студентів Пов'язаний університет	Збільшити/зменшити кількість кафедр Додати/видалити спеціальність Збільшити/зменшити кількість студентів	Кількість кафедр = кількість спеціальностей. На кожну спеціальність має припадати ≥ 200 студентів. Кількість спеціальностей залежить від кількості студентів
Сервіс	Технічний посередник: забезпечує обмін даними, введення/виведення, зберігає дані	Формат виводу Шлях до файлу Дані для обробки	Вивести дані на консоль Читати дані з консолі Записати дані у файл Читати дані з файлу	У файл потрібно записати значення атрибутів сутностей, все, що буде виведено на консоль, усі розрахункові значення

Здійснити **об'єктно-орієнтований аналіз** і визначити класи, поля і методи. Спрощена взаємодія між учасниками освітнього середовища: **один університет має один факультет** (модель 1:1). **Сервіс** не є учасником інституційно-адміністративної моделі, є лише інструментом, який використовують інші сутності (університет, факультет) для реалізації своїх дій.

Розробити таблицю зв'язків між сутностями за зразком:

Сутність 1	Сутність 2	Тип зв'язку (1:1)	Опис зв'язку між сутностями

На основі результатів ООА здійснити **об'єктно-орієнтоване проєктування** і побудувати діаграму класів. На основі діаграми класів розробити **код класів** мовою C# для обробки даних про кафедру і студента. Розробити **сценарій роботи програми** і побудувати Use Case діаграму.
На основі Use Case діаграми написати код для реалізації сценарію роботи програми у функції Main() програмного моделювання предметної області «Інституційно-адміністративна модель».

Версія 1 (пункти завдання 1,2,3,4,5,6).

1. Створити класи **University, Faculty, Service**. Кожен клас має бути створений в окремому файлі командою *Project → Add class.* Визначити в класах закриті поля, що відповідають атрибутам предметної област (табл.5.1).
2. Визначити в класах конструктор за замовчуванням (без параметрів для ініціалізації полів класів нульовими та пустими (для типу *string*) значеннями).
3. Визначити в класах конструктори з параметрами для ініціалізації полів класів початковими значеннями та конструктор копії для створення нових об'єктів як копії існуючого об'єкта.
4. Визначити в класах відкриті властивості (get, set) для доступу до закритих полів та зміни значень ініціалізованих в конструкторах полів класів.
5. Визначити в класах відкриті методи для реалізації поведінки об'єктів через зміну їх станів відповідно до табл.5.1.

Алгоритм розрахунку рейтингу університету за версією Osvita.ua (https://osvita.ua/vnz/rating/25752/): консолідований бал університету визначається як сума балів за рейтингами «SCOPUS», «ТОП200 Україна», «Бал ЗНО на контракт». Враховуються показники: кількість наукових праць, студентів, міжнародних проєктів, ЗНО вступників по кожному факультету, що є ефективністю факультету .
*Рейтинг = f(середній бал ЗНО, кількість публікацій, кількість студентів, кількість факультетів * середня ефективність усіх факультетів)*
Бали за різні номінації отримати в результаті генерації псевдовипадкових цілих чисел в діапазоні від 0 до 200 методами класу Random або взяти реальні.

Алгоритм розрахунку розміру річного фінансування університет: здійснюється відповідно до рейтингу університету та кількості студентів, враховуючи, що підготовка одного бюджетного студента в рік коштує від $2000 до $6000. Якщо університет у **топ-10,** він отримує **100% базового фінансування**. Якщо нижче — коефіцієнт пропорційно знижується.
*Фінансування = Базовий_фонд * Коефіцієнт_рейтингу – (Адміністративні_витрати + Кількість_факультетів * умовна_витрата_на_структуру),*
де *Базовий_фонд* - загальний обсяг коштів, який може бути виділений університету з держбюджету або спонсорських джерел; *Коефіцієнт_рейтингу* - число (0.0 – 1.0 або 0% – 100%), що визначається на основі рейтингу університету; *Адміністративні_витрати* (20 - 30% від бюджету університету) - постійні витрати на управлінський апарат, *Кількість_факультетів* визначається в процесі роботи програми; умовна_витрата_на_структуру - середня сума, яка витрачається на забезпечення роботи одного факультету.

6. У функції Main() реалізувати сценарій роботи програми відповідно до Use Case діаграми.

Версія 2 (Версія 1 + п.7) в окремому проєкті

7. Додати до класу **Faculty** вбудований (вкладений) клас Стартап_Інкубатор (**StartupIncubator**) факультету як додаткову сутність згідно з табл. 5.2:

Таблиця 5.2. Додаткові сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умови виконання дій
Стартап-Інкубатор	Інституція підтримки інноваційної активності студентів	Назва інкубатора, Кількість менторів, Кількість активних проєктів, Кількість залучених студентів, Список стартапів Обсяг інвестицій, які інкубатор отримає для впровадження своїх старапів	Додати студентський стартап Призначити ментора Оцінити результативність Розрахувати рейтинг результативності Вибрати найкращий стартап проєкт.	

Алгоритм методу «Вибрати найкращий стартап проєкт». Вважаємо, що є 10 проектів і 5 експертів, кожний з яких ставить проєкту оцінку. В якості оцінок проєкту виступають псевдовипадкові числа в діапазоні від 1 до 10. В результаті отримаємо матрицю вимірністю 5*10 (кількість експертів (рядки)*кількість проєктів (стовпці)). Для кожного проєкту знаходимо суму балів (сума елементів по стовпчиках), потім розраховуємо середнє арифметичний бал для проєкту, поділивши сумарний бал проєкту на кількість експертів. Сортуємо середні бали за зростанням. Проєкт, який має найменший середній бал, вважатимемо найкращим.

Алгоритм методу «Рейтинг результативності студентів в Стартап-Інкубаторе». Для кожного студента визначаємо долю (значення від 0 до 1) його участі в кожному проєкті за допомогою генератору псевдовипадкових чисел. Отриману долю участі множимо на обсяг інвестицій, сортуємо отримані значення за спаданням. Студент з найвищим показником є найрезультативнішим.

Версія 3 (Версія 2 + п.8) в окремому проєкті.

8. Модифікувати клас **University**, подавши його як частковий, поділивши клас на дві частини кожна в окремому файлі: в один файл включити метод розрахунок рейтингу університету, в інший файл – метод розрахунку розміру річного фінансування університету.

Версія 4 (Версія 3 + п.9) в окремому проєкті.

9. Додати до проекту новий статичний клас **StartupProject** (Стартап проєкт), включивши в нього функцію (на вибір студента) з варіанта 5 лабораторної роботи 2, зробивши її статичною.

Алгоритми методів, які не описані в завданні, студент розробляє самостійно.

6.

Опис предметної області. Існує освітнє середовище, учасниками якого є кафедра і ІТ-компанія. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних участникам освітнього середовища операцій. В таблиці 6.1 подані ролі, атрибути та операції сутностей освітнього середовища.

Таблиця 6.1. Основні сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності	Умова виконання дії
Кафедра	Освітній підрозділ ЗВО	Назва Якість освітньої програми Кількість викладачів Кількість дисциплін Кількість креативних студентів Кількість реальних проєктів	Покращити якість освітньої програми Залучити фахівців з компанії Збільшити кількість креативних студентів	Студент бере участь у ≥ 1 реальному проєкті. Студент створив хоча б 1 власну розробку / стартап. Студент отримав ≥ 90 балів за інноваційні курси. Студент бере участь у наукових дослідженнях, хакатонах, олімпіадах
ІТ-компанія	Роботодавець, партнер кафедри, джерело практичного досвіду.	Назва Кількість співробітників Прибуток Прийнято випускників на роботу Реальний проєкт	Найняти випускників Збільшити прибуток	
Сервіс	Технічний посередник: забезпечує обмін даними, введення/виведення даних, зберігає дані	Формат виводу Шлях до файлу Дані для обробки	Вивести дані на консоль Читати дані з консолі Записати дані у файл Читати дані з файлу	У файл потрібно записати значення атрибутів сутностей, все, що буде виведено на консоль, усі розрахункові значення

Здійснити **об'єктно-орієнтований аналіз** і визначити класи, поля і методи. Спрощена взаємодія між учасниками освітнього середовища: **одна кафедра співпрацює з однією ІТ-компанією** (модель 1:1). **Сервіс** не є учасником освітнього середовища, є лише інструментом, який використовують інші сутності (кафедра, ІТ-компанія) для реалізації своїх дій.

Розробити таблицю зв'язків між сутностями за зразком:

Сутність 1	Сутність 2	Тип зв'язку (1:1)	Опис зв'язку між сутностями

На основі результатів ООА здійснити **об'єктно-орієнтоване проєктування** і побудувати діаграму класів.

На основі діаграми класів розробити **код класів** мовою C# для обробки даних про кафедру і студента.

Розробити **сценарій роботи програми** і побудувати Use Case діаграму.

На основі Use Case діаграми написати код для реалізації сценарію роботи програми у функції Main() програмного моделювання предметної області «Інституційно-адміністративна модель».

Версія 1 (пункти завдання 1,2,3,4,5,6).

1. Створити класи **Department, Company** та **Service**. Кожен клас має бути створений в окремому файлі командою *Project → Add class*. Визначити в класах закриті поля, що відповідають атрибутам предметної област (табл.6.1).
2. Визначити в класах конструктор за замовчуванням (без параметрів для ініціалізації полів класів нульовими та пустими (для типу *string*) значеннями).
3. Визначити в класах конструктори з параметрами для ініціалізації полів класів початковими значеннями та конструктор копії для створення нових об'єктів як копії існуючого об'єкта.

4. Визначити в класах відкриті властивості (get, set) для доступу до закритих полів та зміни значень ініціалізованих в конструкторах полів класів.
5. Визначити в класах відкриті методи для реалізації поведінки об'єктів через зміну їх станів відповідно до табл.6.1.

Алгоритм методу «Покращити якість освітньої програми». Якщо якість освітньої програми не відповідає вимогам (наприклад, <100%), залучити фахівців з компанії, додати нові дисципліни, оновити силабуси дисциплін, збільшити кількість викладачів, залучити студентів до виконання проєктів, збільшити якість освітньої програми на кілька відсотків.

Алгоритм методу «Збільшити кількість креативних студентів». Визначити поточний рівень креативності студентів (порахувати кількість студентів, які беруть участь у реальних проєктах, створили власну розробку/стартап, отримали ≥ 90 балів за інноваційні курси, беруть участь у наукових дослідженнях, олімпіадах**).** Визначити проєкти для залучення студентів, задати кількість студентських нагород. Розрахувати нову кількість креативних студентів.

Алгоритм методу «Збільшити прибуток компанії». Розрахувати прибуток: *Додатковий прибуток = Кількість співробітників * коефіцієнт,* де *коефіцієнт* - це множник, який відображає внесок одного співробітника у прибуток компанії (наприклад, згенерувати число від 1000 до 20000, що означатиме, в середньому кількість грн в місяць прибутку від одного працівника). Додати додатковий прибуток до загального прибутку компанії. Зафіксувати нове значення прибутку (запис у файл).

6. У функції Main() реалізувати сценарій роботи програми відповідно до Use Case діаграми.

Версія 2 (Версія 1 + п.7) в окремому проєкті

7. Додати до класу **Company** вбудований (вкладений) клас IT акселератор (**ITAccelerator**) як додаткову сутність згідно з табл. 6.2:

Таблиця 6.2. Додаткові сутності предметної області

Сутність	Роль сутності	Атрибути сутності	Операції (дії) сутності
IT акселератор	Сприяти розвитку стартапів, залучати студентів та випускників до підприємницької діяльності	Назва Кількість проектів, участь в яких беруть викладачі і студенти; Кількість студентів, які навчаються в акселераторі; Обсяг інвестицій, які акселератор передає на кафедру. Рівень інноваційності	Додати студентський стартап Призначити ментора Оцінити результативність стартапу Розрахувати рейтинг студентів для конкурсу в акселератор; Вибрати найкращий стартап проєкт.

Алгоритм методу «Вибрати найкращий стартап проєкт». Вважаємо, що є 10 проектів і 5 експертів, кожний з яких ставить проєкту оцінку. В якості оцінок проєкту виступають псевдовипадкові числа в діапазоні від 1 до 10. В результаті отримаємо матрицю вимірністю 5*10 (кількість експертів (рядки)*кількість проєктів (стовпці)). Для кожного проєкту знаходимо суму балів (сума елементів по стовпчиках), потім розраховуємо середнє арифметичний бал для проєкту, поділивши сумарний бал проєкту на кількість експертів. Сортуємо середні бали за зростанням. Проєкт, який має найменший середній бал, вважатимемо найкращим.

Алгоритм методу «Рейтинг результативності студентів в Стартап-Інкубаторі». Для кожного студента визначаємо долю (значення від 0 до 1) його участі в кожному проєкті за допомогою генератору псевдовипадкових чисел. Отриману долю участі множимо на обсяг інвестицій, сортуємо отримані значення за спаданням. Студент з найвищим показником є найрезультативнішим.

Версія 3 (Версія 2 + п.8) в окремому проєкті.

8. Модифікувати клас **Department**, подавши його як частковий, поділивши клас на дві частини кожна в окремому файлі: в один файл включити метод покращення якості освітньої програми, в інший файл – метод розрахунку обсягу інвестицій на кафедру від стартапів: *Інвестиції на кафедру* = ∑ *(Інвестиції від і-го стартапу* × *Коефіцієнт_повернення),* де *Інвестиції від і-го стартапу* = сумі коштів, залучених і-им стартапом, коефіцієнт повернення — це відсоток від прибутку компанії, який йде на підтримку освітньої бази кафедри.

Версія 4 (Версія 3 + п.9) в окремому проєкті.

9. Додати до проекту новий статичний клас **StartupProject** (Стартап проєкт), включивши в нього функцію (на вибір студента) з варіанта 6 лабораторної роботи 2, зробивши її статичною.

Алгоритми методів, які не описані в завданні, студент розробляє самостійно.

7.

Опис предметної області. Існує освітнє середовище, учасниками якого є студент та його індивідуальна навчальна програма. Застосування SOLID принципу єдиної відповідальності вимагає включення в систему сутності Сервіс для реалізації не притаманних учасникам освітнього середовища операцій. В таблиці 7.1 подані ролі, атрибути та операції сутностей освітнього середовища.