

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА**

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

**до виконання практичних та самостійних робіт
із навчальної дисципліни**

«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

*(для здобувачів першого (бакалаврського) рівня вищої освіти
денної форми навчання зі спеціальностей F3, 122 – Комп'ютерні науки)*

Харків

ХНУМГ ім. О. М. Бекетова

2025

Методичні рекомендації до виконання практичних та самостійних робіт із навчальної дисципліни «Об'єктно-орієнтоване програмування» (для здобувачів першого (бакалаврського) рівня вищої освіти денної форми навчання зі спеціальностей F3, 122 – Комп'ютерні науки) / Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова ; уклад. В. М. Бредіхін. – Харків : ХНУМГ ім. О. М. Бекетова, 2025. – 58 с.

Укладач канд. техн. наук, доц. В. М. Бредіхін

Рецензент

О. Б. Костенко, кандидат фізико-математичних наук, доцент кафедри комп'ютерних наук та інформаційних технологій Харківського національного університету міського господарства ім. О. М. Бекетова

Рекомендовано кафедрою комп'ютерних наук та інформаційних технологій, протокол № 17 від 26.06.2024

ЗМІСТ

ВСТУП.....	4
Практична робота № 1 Основні поняття ООП. Класи та об'єкти. Функції доступу. Вказівник this	5
Практична робота № 2 Конструктори	16
Практична робота № 3 Успадкування, агрегація та композиція	19
Практична робота № 4 Поліморфізм. Обробка винятків	30
Практична робота № 5 Шаблони в C#.....	41
Практична робота № 6 Контейнерні класи. Стандартна бібліотека шаблонів (STL) в C#	49
Список використаних джерел	55

ВСТУП

Мета дисципліни — засвоєння базових знань з основ об'єктно-орієнтованого програмування, включно з основними поняттями, парадигмами та принципами об'єктно-орієнтованого програмування. Оволодіння базовими навичками проєктування програмних систем, роботи з найбільш вживаними шаблонами проєктування, набуття навичок об'єктно-орієнтованого програмування та оволодіння мовою програмування C#.

Завдання — набуття знань, умінь та навичок (компетенцій) на рівні новітніх досягнень у програмуванні, відповідно до кваліфікації «фахівець з інформаційних технологій».

У результаті вивчення навчальної дисципліни студент повинен:

знати: засоби проєктування систем; стандарти моделювання, методи аналізу потреб; методи розробки програмного забезпечення; принципи проєктування користувацьких інтерфейсів; інструментальні засоби для формалізації функціональних специфікацій; мови програмування; основні процеси, фази та ітерації життєвого циклу програмного забезпечення; технології та методи проєктування та програмування; ефективні підходи щодо проєктування програмного забезпечення;

вміти: розробляти шаблони і моделі проєктування; збирати, формалізувати і оцінювати функціональні та інші вимоги; розробляти специфікації з урахуванням встановлених вимог; роз'яснювати і подавати проєкти / розробки замовникам; використовувати моделі даних; розробляти ПЗ на основі моделі; розробляти ПЗ із використанням сучасних технологій розробки; складати план тестування і контролювати його виконання; демонструвати процеси та результати професійної діяльності, розробляючи презентації, звіти; використовувати методи компонентної розробки програмного забезпечення, виокремлюючи інтерфейси і реалізації та взаємодію між модулями, підсистемами і компонентами.

Практична робота № 1

Основні поняття ООП. Класи та об'єкти. Функції доступу.

Вказівник this

Мета: ознайомитись з основними поняттями ООП. Вивчити поняття «клас», «об'єкт», «сеттер», «геттер» та навчитись їх програмно реалізовувати мовою C#.

Теоретична частина

Клас – це конструкція в ООП, що є шаблоном або описом структури та поведінки об'єктів. Клас є абстракцією, яка групує дані (властивості) та функції (методи), що пов'язані між собою логічно. Він визначає набір атрибутів (членів класу), які представляють стан об'єктів цього класу, а також методи, які визначають операції, які можуть бути виконані над цими об'єктами.

Класи визначаються за допомогою ключового слова `class` та мають назву, за допомогою якої їх можна ідентифікувати. Клас може мати конструктори, які використовуються для ініціалізації об'єктів класу, а також може містити статичні поля та методи, які використовуються для спільного використання даних між усіма об'єктами цього класу.

Об'єкт – це конкретний екземпляр класу. Він створюється на основі класу шляхом процесу, який називається інстанціюванням або створенням об'єкта. Кожен об'єкт має свій власний набір значень властивостей, які відображають його стан, та може виконувати методи, що визначені в класі.

Об'єкти зберігають свій внутрішній стан у вигляді значень властивостей, які можуть бути унікальними для кожного об'єкта. Методи об'єктів – це операції або функції, які можуть бути виконані над цими об'єктами.

Об'єкти взаємодіють один з одним шляхом виклику методів та обміну даними.

Основна ідея ООП полягає в організації програми як набору взаємодіючих об'єктів, які співпрацюють для вирішення певних задач. Класи визначають загальну структуру та поведінку, а об'єкти є конкретними представниками цих класів, здатними зберігати стан та виконувати дії, визначені класом.

Приклад:

```
using System;
```

```
namespace CarApp
{
    // Клас «Автомобіль»
    class Car
    {
        public string Brand;
        public string Model;
        public int Year;

        public void StartEngine()
```

```

    {
        Console.WriteLine(«Двигун автомобіля запущено!»);
    }

    public void StopEngine()
    {
        Console.WriteLine(«Двигун автомобіля зупинено!»);
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Створення об'єкта класу «Автомобіль»
        Car myCar = new Car();

        // Налаштування властивостей об'єкта
        myCar.Brand = «BMW»;
        myCar.Model = «X5»;
        myCar.Year = 2022;

        // Виведення інформації про автомобіль
        Console.WriteLine(«Марка автомобіля: » + myCar.Brand);
        Console.WriteLine(«Модель автомобіля: » + myCar.Model);
        Console.WriteLine(«Рік випуску автомобіля: » + myCar.Year);

        // Виклик методів об'єкта
        myCar.StartEngine(); // Запуск двигуна
        myCar.StopEngine();  // Зупинка двигуна
    }
}

```

У цьому прикладі ми оголосили клас «Car», який представляє автомобіль. Він має властивості – «brand», «model» та «year», а також методи «startEngine()» та «stopEngine()», які відповідають за запуск та зупинку двигуна.

У функції «main()» ми створюємо об'єкт «myCar» класу «Car». Потім ми налаштовуємо властивості об'єкта, використовуючи оператор крапки (.), і викликаємо методи об'єкта.

В результаті на консоль будуть виведені властивості автомобіля (марка, модель, рік випуску), а також повідомлення про запуск та зупинку двигуна.

Функції доступу (сеттери та геттери) є методами, що використовуються в об'єктно-орієнтованому програмуванні (ООП) для отримання (читання) та встановлення (запису) значень приватних членів класу. Ці методи надають контроль доступу до даних класу та допомагають забезпечити принцип інкапсуляції. Це означає, що дані класу знаходяться внутрішньо і мають обмежений доступ ззовні.

Геттери (getter methods) – це методи, які використовуються для отримання (читання) значень приватних членів класу. Вони зазвичай

повертають значення цих членів і мають повернутий тип, що відповідає типу члена класу. Геттери дозволяють зовнішньому коду отримувати доступ до значень приватних даних класу без безпосереднього доступу до них. Вони можуть бути корисними для отримання значень для подальшого використання або виведення на екран.

Наприклад, у класі «Person» може бути приватний член «name», і для отримання цього значення може бути створений «геттер»:

```
using System;

class Person
{
    private string name;

    // Геттер для отримання значення name
    public string GetName()
    {
        return name;
    }

    // (Додатково) Сеттер для встановлення значення name
    public void SetName(string value)
    {
        name = value;
    }
}
```

Сеттери (setter methods) – це методи, які використовуються для встановлення (запису) значень приватних членів класу. Вони обирають параметр з новим значенням і встановлюють це значення для відповідного приватного члена класу. Сеттери дозволяють контролювати доступ до приватних даних класу і забезпечувати перевірку або обробку значень перед їх збереженням.

Продовжуючи приклад з класом «Person», можна створити сеттер для встановлення нового значення для «name»:

```
using System;

class Person
{
    private string name;

    // Геттер для отримання значення name
    public string GetName()
    {
        return name;
    }

    // Сеттер для встановлення значення name
    public void SetName(string newName)
```

```

    {
        name = newName;
    }
}

class Program
{
    static void Main()
    {
        Person person = new Person();

        person.SetName(«Олександр»);
        Console.WriteLine(«Ім'я: « + person.GetName());
    }
}

```

За допомогою сеттера зовнішній код може встановити нове значення для «name», і внутрішній код класу може здійснити додаткову перевірку або обробку цього значення перед збереженням.

Використання геттерів та сеттерів дозволяє забезпечити кращий контроль над доступом до приватних даних класу, зберігаючи їх недоступними безпосередньо ззовні класу і забезпечуючи правильність операцій з цими даними.

Ключове слово «this» в об'єктно-орієнтованому програмуванні (ООП) вказує на поточний об'єкт, із яким пов'язаний виклик методу або доступ до членів класу. Воно уособлює вказівник на сам об'єкт, на якому викликається метод або з якого відбувається доступ до членів класу.

Ключове слово «this» може використовуватись в контексті класу для посилення на його власні члени (змінні та методи). Це особливо корисно, коли ім'я параметра методу або локальної змінної конфліктує з іменем члена класу. Використання «this» дозволяє уникнути неоднозначностей і вказати на конкретний член класу.

Наприклад, розглянемо клас «Person» з методом «setName», який встановлює значення приватного члена «name»:

```

using System;

class Person
{
    private string name;

    public void SetName(string name)
    {
        this.name = name;
    }

    // Для демонстрації (не обов'язково)
    public void PrintName()
    {

```

```

        Console.WriteLine(«Ім'я: » + name);
    }
}

class Program
{
    static void Main()
    {
        Person person = new Person();
        person.SetName(«Олександр»);
        person.PrintName();
    }
}

```

У цьому прикладі параметр методу «setName» має таке ж ім'я, як і приватний член «name». Використання «this.name» дозволяє чітко вказати, що ми звертаємось саме до члена класу «name», а не до параметра методу.

Крім того, ключове слово «this» може бути корисним у разі передавання посилання на поточний об'єкт як аргумент у виклику методу або під час створення ланцюжка методів.

Загалом, «this» використовується для доступу до членів класу зсередини самого класу і допомагає уникнути конфліктів імен, які можуть виникати між параметрами методів та членами класу.

Завдання

Варіант 1

1. Напишіть клас «Book» для подання книги. Клас повинен мати такі властивості та функціональність:

1) приватні поля класу:

- «title» (назва книги);
- «author» (автор книги);
- «year» (рік видання книги);

2) публічні методи класу:

- метод «setTitle()», який дозволяє задати назву книги;
- метод «getTitle()», який повертає назву книги;
- метод «setAuthor()», який дозволяє задати автора книги;
- метод «getAuthor()», який повертає автора книги;
- метод «setYear()», який дозволяє задати рік видання книги;
- метод «getYear()», який повертає рік видання книги;

2. Створіть об'єкт класу «Book».

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про книгу на екран, використовуючи методи доступу до полів.

5. Реалізуйте програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 2

1. Напишіть клас «Rectangle» для представлення прямокутника.
 2. Клас повинен мати такі властивості та функціональність:
 - 1) приватні поля класу:
 - «width» (ширина прямокутника);
 - «height» (висота прямокутника);
 - 2) публічні методи класу:
 - метод «setWidth()», який дозволяє задати значення ширини прямокутника;
 - метод «setHeight()», який дозволяє задати значення висоти прямокутника;
 - метод «getArea()», який повертає площу прямокутника (обчислюється як добуток ширини та висоти);
 - метод «getPerimeter()», який повертає периметр прямокутника (обчислюється як сума усіх сторін прямокутника).
 3. Створіть об'єкт класу «Rectangle».
 4. Задайте значення ширини та висоти прямокутника.
 5. Викличте методи «getArea()» та «getPerimeter()» для отримання площі та периметра прямокутника.
 6. Виведіть результати на екран.
 7. Реалізуйте програму за допомогою роздільної компіляції.
- У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 3

1. Напишіть клас «BankAccount» для керування банківським рахунком. Клас повинен мати такі властивості та функціональність:
 - 1) приватні поля класу:
 - «balance» (баланс рахунку, типу «double»);
 - «accountNumber» (номер рахунку, типу «string»);
 - «ownerName» (ім'я власника рахунку, типу string);
 - 2) публічні методи класу:
 - метод «setBalance()», який дозволяє задати початковий баланс рахунку;
 - метод «getBalance()», який повертає поточний баланс рахунку;
 - метод «setAccountNumber()», який дозволяє задати номер рахунку;
 - метод «getAccountNumber()», який повертає номер рахунку;
 - метод «setOwnerName()», який дозволяє задати ім'я власника рахунку;
 - метод «getOwnerName()», який повертає ім'я власника рахунку;
 - метод «deposit()», який дозволяє здійснити поповнення рахунку на певну суму;
 - метод «withdraw()», який дозволяє зняти гроші з рахунку на певну суму.
2. Створіть об'єкт класу «BankAccount».

3. Задайте значення полів об'єкта, включно з балансом рахунку, номером рахунку та ім'ям власника.

4. Використовуйте методи для отримання та зміни значень полів об'єкта.

5. Викличте методи для поповнення та зняття грошей з рахунку.

6. Виведіть інформацію про рахунок на екран.

7. Реалізуйте програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 4

1. Створіть клас «Person» (людина), який має такі властивості та функціональність:

1) приватні поля класу:

- «name» (ім'я людини);
- «age» (вік людини);
- «address» (адреса людини).

2) публічні методи класу:

- метод «setname()», який дозволяє задати ім'я людини;
- метод «getname()», який повертає ім'я людини;
- метод «setage()», який дозволяє задати вік людини;
- метод «getage()», який повертає вік людини;
- метод «setaddress()», який дозволяє задати адресу людини;
- метод «getaddress()», який повертає адресу людини.

2. Створіть об'єкт класу «Person».

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про людину на екран, використовуючи методи для отримання значень полів.

5. Реалізуйте програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 5

1. Створіть клас «Employee» (співробітник), який має такі властивості та функціональність:

1) приватні поля класу:

- «name» (ім'я співробітника);
- «id» (ідентифікатор співробітника);
- «salary» (заробітна плата співробітника);

2) публічні методи класу:

- метод «setname()», який дозволяє задати ім'я співробітника;
- метод «getname()», який повертає ім'я співробітника;
- метод «setid()», який дозволяє задати ідентифікатор співробітника;
- метод «getid()», який повертає ідентифікатор співробітника;

- метод «setsalary()», який дозволяє задати заробітну плату співробітника;
 - метод «getsalary()», який повертає заробітну плату співробітника.
2. Створіть об'єкт класу «Employee».
 3. Задайте значення полів об'єкта за допомогою відповідних методів.
 4. Виведіть інформацію про співробітника на екран, використовуючи методи для отримання значень полів.
 5. Реалізуйте програму за допомогою роздільної компіляції.
- У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 6

1. Створіть клас «Student» (студент), який має такі властивості та функціональність:
 - 1) приватні поля класу:
 - «name» (ім'я студента);
 - «age» (вік студента);
 - «major» (спеціальність студента);
 - 2) публічні методи класу:
 - метод «setname()», який дозволяє задати ім'я студента;
 - метод «getname()», який повертає ім'я студента;
 - метод «setage()», який дозволяє задати вік студента;
 - метод «getage()», який повертає вік студента;
 - метод «setmajor()», який дозволяє задати спеціальність студента;
 - метод «getmajor()», який повертає спеціальність студента.
 2. Створіть об'єкт класу «Student».
 3. Задайте значення полів об'єкта за допомогою відповідних методів.
 4. Виведіть інформацію про студента на екран, використовуючи методи для отримання значень полів.
 5. Реалізуйте програму за допомогою роздільної компіляції.
- У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 7

1. Створіть клас «Animal» (тварина), який має такі властивості та функціональність:
 - 1) приватні поля класу:
 - «name» (ім'я тварини);
 - «species» (вид тварини);
 - «age» (вік тварини);
 - 2) публічні методи класу:
 - метод setname(), який дозволяє задати ім'я тварини;
 - метод getname(), який повертає ім'я тварини;
 - метод setspecies(), який дозволяє задати вид тварини;
 - метод getspecies(), який повертає вид тварини;

- метод `setage()`, який дозволяє задати вік тварини;
 - метод `getage()`, який повертає вік тварини.
 - 2. Створіть об'єкт класу «Animal».
 - 3. Використайте метод «`setName()`» для задання імені тварини.
 - 4. Використайте метод «`getName()`» для отримання імені тварини та виведіть його на екран.
 - 5. Використайте метод «`setSpecies()`» для задання виду тварини.
 - 6. Використайте метод «`getSpecies()`» для отримання виду тварини та виведіть його на екран.
 - 7. Використайте метод «`setAge()`» для задання віку тварини.
 - 8. Використайте метод «`getAge()`» для отримання віку тварини та виведіть його на екран.
 - 9. Реалізуйте програму за допомогою роздільної компіляції.
- У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 8

1. Створіть клас «FootballPlayer» (футболіст), який має такі властивості та функціональність:
 - 1) приватні поля класу:
 - «`name`» (ім'я футболіста);
 - «`age`» (вік футболіста);
 - «`position`» (позиція на полі);
 - 2) публічні методи класу:
 - метод «`setname()`», який дозволяє задати ім'я футболіста;
 - метод «`getname()`», який повертає ім'я футболіста;
 - метод «`setage()`», який дозволяє задати вік футболіста;
 - метод «`getage()`», який повертає вік футболіста;
 - метод «`setposition()`», який дозволяє задати позицію футболіста;
 - метод «`getposition()`», який повертає позицію футболіста.
 2. Створіть об'єкт класу «FootballPlayer».
 3. Використайте метод «`setName()`» для задання імені футболіста.
 4. Використайте метод «`getName()`» для отримання імені футболіста та виведіть його на екран.
 5. Використайте метод «`setAge()`» для задання віку футболіста.
 6. Використайте метод «`getAge()`» для отримання віку футболіста та виведіть його на екран.
 7. Використайте метод «`setPosition()`» для задання позиції футболіста.
 8. Використайте метод «`getPosition()`» для отримання позиції футболіста та виведіть її на екран.
 9. Реалізуйте програму за допомогою роздільної компіляції.
- У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними

Варіант 9

1. Створіть клас «Movie» (фільм), який має такі властивості та функціональність:

1) приватні поля класу:

- «title» (назва фільму);
- «director» (режисер фільму);
- «year» (рік випуску фільму);
- «duration» (тривалість фільму в хвилинали);

2) публічні методи класу:

- метод «settitle()», який дозволяє задати назву фільму;
- метод «gettitle()», який повертає назву фільму;
- метод «setdirector()», який дозволяє задати режисера фільму;
- метод «getdirector()», який повертає режисера фільму;
- метод «setyear()», який дозволяє задати рік випуску фільму;
- метод «getyear()», який повертає рік випуску фільму;
- метод «setduration()», який дозволяє задати тривалість фільму;
- метод «getduration()», який повертає тривалість фільму.

2. Створіть об'єкт класу «Movie».

3. Використайте метод «setTitle()» для задання назви фільму.

4. Використайте метод «getTitle()» для отримання назви фільму та виведіть її на екран.

5. Використайте метод «setDirector()» для задання режисера фільму.

6. Використайте метод «getDirector()» для отримання режисера фільму та виведіть його на екран.

7. Використайте метод «setYear()» для задання року випуску фільму.

8. Використайте метод «getYear()» для отримання року випуску фільму та виведіть його на екран.

9. Використайте метод «setDuration()» для задання тривалості фільму.

10. Використайте метод «getDuration()» для отримання тривалості фільму та виведіть її на екран.

11. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 10

1. Створіть клас «TravelDestination» (туристичний напрямок), який має такі властивості та функціональність:

1) приватні поля класу:

- «name» (назва туристичного напрямку);
- «country» (країна, в якій знаходиться напрямок);
- «rating» (рейтинг популярності напрямку);

2) публічні методи класу:

- метод «setname()», який дозволяє задати назву туристичного напрямку;
- метод «getname()», який повертає назву туристичного напрямку;

– метод «setcountry()», який дозволяє задати країну, в якій знаходиться напрямок;

– метод «getcountry()», який повертає країну, в якій знаходиться напрямок;

– метод «setrating()», який дозволяє задати рейтинг популярності напрямку;

– метод «getrating()», який повертає рейтинг популярності напрямку.

2. Створіть об'єкт класу «TravelDestination».

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про туристичний напрямок на екран, використовуючи методи «getName()», «getCountry()» та «getRating()».

5. Змініть значення рейтингу популярності напрямку за допомогою методу «setRating()».

6. Виведіть оновлену інформацію про туристичний напрямок на екран, знову використовуючи методи «getName()», «getCountry()» та «getRating()».

7. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Практична робота № 2

Конструктори

Мета: ознайомитись з основними поняттями «конструктор» і «деструктор» в ООП та навчитись їх програмно реалізовувати мовою С#.

Теоретична частина

Конструктор є спеціальним методом в класі, який викликається автоматично під час створення нового об'єкта цього класу. Його основна функція полягає в ініціалізації об'єкта, тобто встановленні його початкового стану.

У мові С# конструктор має те саме ім'я, що й клас, і не має повертального значення (навіть не вказується тип «void»). Конструктор може мати параметри, але також може бути й без параметрів. Зазвичай використовуються параметри конструктора для передавання значень до членів даних «об'єкта».

Основні види конструкторів

Конструктор за замовчуванням (default constructor) – це конструктор без параметрів, який викликається під час створення об'єкта без передачі будь-яких значень. Він може ініціалізувати члени даних об'єкта значеннями за замовчуванням.

Приклад:

using System;

```
class MyClass
{
    // Конструктор за замовчуванням
    public MyClass()
    {
        // Виконання початкових операцій
        Console.WriteLine(«Виконання конструктора за замовчуванням»);
    }
}

class Program
{
    static void Main()
    {
        MyClass obj = new MyClass();
    }
}
```

Параметризований конструктор (parameterized constructor) – це конструктор, який обирає параметри і використовує їх для ініціалізації членів даних об'єкта.

Приклад:

```
using System;
```

```
class Point
{
    private int x, y;

    // Конструктор з параметрами
    public Point(int xCoord, int yCoord)
    {
        x = xCoord;
        y = yCoord;
    }

    // Для демонстрації координат (необов'язково)
    public void Print()
    {
        Console.WriteLine($ «Координати точки: ({x}, {y})»);
    }
}

class Program
{
    static void Main()
    {
        Point p = new Point(3, 5);
        p.Print(); // Виведе: Координати точки: (3, 5)
    }
}
```

Конструктор копіювання (copy constructor) – це конструктор, який обирає об'єкт того ж класу, як параметр і створює новий об'єкт, який є копією цього переданого об'єкта. Конструктор копіювання використовується, коли потрібно зробити копію об'єкта для подальшої роботи з ним, а не просто посилання на нього.

Приклад:

```
using System;
```

```
class MyClass
{
    public int Data; // Для прикладу, деякі дані

    // Конструктор за замовчуванням
    public MyClass()
    {
        Data = 0;
    }

    // Конструктор копіювання
    public MyClass(MyClass other)
```

```

    {
        // Виконання операцій копіювання
        Data = other.Data;
    }

    public void Print()
    {
        Console.WriteLine(«Значення Data: » + Data);
    }
}

class Program
{
    static void Main()
    {
        MyClass original = new MyClass();
        original.Data = 42;

        MyClass copy = new MyClass(original); // Виклик конструктора копіювання

        copy.Print(); // Виведе: значення Data: 42
    }
}

```

Конструктори дозволяють забезпечити коректну ініціалізацію об'єктів і можуть бути потужним інструментом у процесі програмування на C#.

Завдання

1. Реалізуйте конструктор за замовчуванням, конструктор з параметрами та конструктор копіювання для вашого класу. Конструктор за замовчуванням має встановлювати значення полів.
2. Конструктор з параметрами має обирати значення для кожного поля.
3. Конструктор копіювання має копіювати значення полів з іншого об'єкта класу.
4. Реалізуйте деструктор для класу. Деструктор має виводити повідомлення про знищення об'єкта.
5. У функції «main» створіть об'єкт за допомогою конструктора за замовчуванням та виведіть значення його полів.
6. Створіть новий об'єкт за допомогою конструктора з параметрами та встановіть значення для полів. Виведіть значення полів цього об'єкта.
7. Створіть ще один об'єкт і скопіюйте значення полів з першого об'єкта за допомогою конструктора копіювання. Виведіть значення полів цього об'єкта.
8. Завершіть функцію «main», що призведе до виходу з області видимості створених об'єктів і виклику їхніх деструкторів. Переконайтесь, що повідомлення про знищення об'єктів виводяться.

Практична робота № 3

Успадкування, агрегація та композиція

Мета: ознайомитись з основними поняттями – «успадкування», «агрегація» та «композиція» в ООП та навчитись їх програмно реалізовувати мовою C#.

Теоретична частина

Агрегація є однією з важливих концепцій в ООП і належить до співвідношення між класами, де один клас є контейнером, що містить об'єкти іншого класу. У випадку агрегації, об'єкти-члени, які належать до контейнера, можуть існувати самостійно і не залежати від контейнера.

Зазвичай агрегація відбувається за допомогою включення (composition) або посилання (reference). У випадку включення, контейнер містить об'єкти-члени як свої прямі члени (наприклад, об'єкти класів або структур), і вони створюються та знищуються разом з контейнером. У випадку посилання, контейнер містить посилання або вказівники на об'єкти-члени, і вони можуть існувати незалежно від контейнера.

Наприклад, давайте розглянемо клас «Автомобіль» і клас «Двигун». У цьому випадку, клас «Автомобіль» може мати агрегаційне відношення з класом «Двигун». Автомобіль може містити об'єкт «Двигун» як свій прямий член, що означає, що двигун створюється та знищується разом з автомобілем. Або ж клас «Автомобіль» може містити посилання на об'єкт «Двигун», що дозволяє розділяти двигун між декількома автомобілями.

Агрегація дозволяє створювати більш складні структури і групувати пов'язані об'єкти разом для зручної організації та керування. Вона також сприяє розподілу відповідальностей і забезпечує зв'язок між класами.

Приклад:

using System;

// Клас «Двигун»

```
class Engine
{
    public void Start()
    {
        Console.WriteLine(«Engine started!»);
    }

    public void Stop()
    {
        Console.WriteLine(«Engine stopped!»);
    }
}
```

// Клас «Автомобіль»

```
class Car
{
```

```

        private Engine engine = new Engine(); // Об'єкт класу «Двигун» як член класу
        «Автомобіль»

        public void StartCar()
        {
            engine.Start();
        }

        public void StopCar()
        {
            engine.Stop();
        }
    }

    class Program
    {
        static void Main()
        {
            Car myCar = new Car();
            myCar.StartCar();
            myCar.StopCar();
        }
    }

```

У цьому прикладі ми маємо клас «Двигун», який має методи «start()» і «stop()» для запуску та зупинки двигуна відповідно. Клас «Автомобіль» агрегує об'єкт класу «Двигун» як свій член.

У головній функції ми створюємо об'єкт «Автомобіль» з назвою «myCar». Викликаючи метод «startCar()», ми викликаємо метод «start()» об'єкта «engine», який запускає двигун. Потім, викликаючи метод «stopCar()», ми викликаємо метод «stop()» об'єкта «engine», який зупиняє двигун.

Композиція – це тип відношення між класами, коли один клас (відомий як контейнер або композит) містить об'єкти іншого класу (відомі як компоненти). У випадку композиції, компоненти не можуть існувати без контейнера, тобто вони залежать від його існування і життєвого циклу.

Основна відмінність між агрегацією і композицією полягає в тому, що в композиції компоненти є прямими членами контейнера і створюються та знищуються разом з ним. Це означає, що коли контейнер створюється або знищується, його компоненти також автоматично створюються або знищуються. Крім того, контейнер відповідає за створення та управління життєвим циклом своїх компонентів.

Давайте розглянемо приклад знову. Припустимо, що у нас є клас «Автомобіль» і клас «Двигун». В цьому випадку, композиційне відношення між ними означає, що клас «Автомобіль» містить об'єкт «Двигун» як свій прямий член. Таким чином, коли створюється об'єкт «Автомобіль», він створює також об'єкт «Двигун». Якщо «Автомобіль» більше не потрібен або видаляється, то разом з ним також буде знищено об'єкт «Двигун».

Композиція дозволяє створювати складніші об'єкти, засновані на ієрархії класів, де один клас повністю залежить від іншого. Вона допомагає керувати взаємозв'язком між компонентами та забезпечує їхню єдність у контексті контейнера.

Приклад:

```
using System;
```

```
// Клас «Серце»
```

```
class Heart
```

```
{
```

```
    public void Beat()
```

```
    {
```

```
        Console.WriteLine(«Heart is beating!»);
```

```
    }
```

```
}
```

```
// Клас «Людина»
```

```
class Person
```

```
{
```

```
    private Heart heart = new Heart(); // Об'єкт класу «Серце» як складова частина класу «Людина»
```

```
    public void DoSomething()
```

```
    {
```

```
        heart.Beat();
```

```
    }
```

```
}
```

```
class Program
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        Person person = new Person();
```

```
        person.DoSomething();
```

```
    }
```

```
}
```

У цьому прикладі ми маємо клас «Серце», який має метод `beat()` для симуляції роботи серця. Клас «Людина» композиційно містить «об'єкт» класу «Серце» як свою частину.

У головній функції ми створюємо об'єкт «Людина» з назвою «person». Викликаючи метод «`doSomething()`», ми викликаємо метод «`beat()`» об'єкта «heart», що відображає роботу серця людини.

Успадкування (іноді його називають також наслідуванням) в об'єктно-орієнтованому програмуванні є одним із фундаментальних концепцій. Воно дозволяє створювати нові класи на основі вже існуючих (батьківських або базових класів), утворюючи ієрархію класів.

Успадкування дозволяє поширювати властивості і поведінку базового класу на похідні класи. Похідні класи отримують доступ до публічних і захищених членів базового класу (таких як поля, методи, властивості) і можуть додавати свої власні члени або перевизначати члени базового класу.

Один з основних принципів успадкування – це утворення спеціалізованих класів на базі загального базового класу. Наприклад, уявімо ієрархію класів «Транспортний засіб» як базовий клас, а «Автомобіль», «Велосипед» та «Мотоцикл» як його похідні класи. У цьому випадку «Транспортний засіб» міститиме загальну функціональність, а похідні класи розширять його функціональність, додаючи власні особливості.

Однією з головних переваг успадкування є полегшення повторного використання коду. Базовий клас може містити загальні функції і властивості, які можуть бути успадковані і використані у всіх похідних класах. Це забезпечує ефективне управління кодом і зменшення дублювання.

Успадкування також сприяє поліморфізму, що дозволяє використовувати об'єкти похідних класів як об'єкти базового класу. Це розширює можливості поліморфного програмування і спрощує роботу зі змінними і колекціями об'єктів.

Наприклад, у мові C# успадкування виконується за допомогою ключового слова «class» або «struct», яке передує іменам похідного класу і базового класу, відокремленим двокрапкою.

Приклад:

using System;

```
// Базовий клас
class BaseClass
{
    // Тіло базового класу
    public void BaseMethod()
    {
        Console.WriteLine(«Метод базового класу»);
    }
}

// Похідний клас
class DerivedClass : BaseClass
{
    // Тіло похідного класу
    public void DerivedMethod()
    {
        Console.WriteLine(«Метод похідного класу»);
    }
}
```

У цьому прикладі клас «DerivedClass» успадковує властивості та методи класу «BaseClass».

Успадкування є важливим інструментом в об'єктно-орієнтованому програмуванні, оскільки дозволяє створювати ієрархії класів з поліморфними можливостями та забезпечує повторне використання коду. Воно сприяє модульності, розширюваності та зрозумілості програмного коду.

Завдання

Варіант 1

1. Створіть клас «Людина» (базовий клас), який містить такі властивості:

- ім'я;
- вік;
- клас «Адреса», який представляє адресу проживання людини (використовуйте композицію).

2. Створіть клас «Студент» (похідний клас від класу «Людина»), який містить додаткові властивості:

- номер студентського квитка;
- курс.

3. Створіть клас «Університет» (базовий клас), який містить такі властивості:

- назву університету;
- список студентів (використовуйте агрегацію).

Реалізуйте методи для додавання та видалення студентів зі списку університету.

4. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Університет», «Людина» та «Студент». Додайте студентів до списку університету та виведіть інформацію про них.

Продемонструйте успадкування, викликавши методи базового класу та похідного класу для об'єктів типу «Студент».

Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 2

1. Створіть клас «Бібліотека», який містить такі властивості:

- назва бібліотеки;
- адреса бібліотеки;
- список книг, які є в наявності в бібліотеці (використовуйте агрегацію).

2. Створіть клас «Бібліотекар», який представляє бібліотекаря, що працює в бібліотеці (використовуйте композицію).

3. Створіть клас «Книга», який містить такі властивості:

- назва книги;
- автор книги;
- рік видання.

4. Створіть клас «Бібліотекар», який містить такі властивості:

- ім'я бібліотекаря;
- посада бібліотекаря.

5. Реалізуйте методи для додавання та видалення книг зі списку книг у бібліотеці.

6. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Бібліотека», «Книга» та «Бібліотекар». Додайте книги до списку книг у бібліотеці та виведіть інформацію про них.

7. Продемонструйте успадкування, створивши похідний клас від «Книга» (наприклад, «Роман») з додатковими властивостями та методами.

8. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 3

1. Створіть клас «Компанія», який містить такі властивості:

- назва компанії;
- адреса компанії;
- список підрозділів, які входять до складу компанії (використовуйте агрегацію);
- головний директор компанії (використовуйте композицію).

2. Створіть клас «Підрозділ», який містить такі властивості:

- назва підрозділу;
- кількість працівників;
- керівник підрозділу.

3. Створіть клас «Працівник», який містить такі властивості:

- прізвище та ім'я працівника;
- посада працівника;
- заробітна плата працівника.

4. Реалізуйте методи для додавання та видалення підрозділів зі списку підрозділів компанії.

5. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Компанія», «Підрозділ» та «Працівник». Додайте підрозділи до списку підрозділів компанії та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від «Працівник» (наприклад, «Менеджер») із додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 4

1. Створіть клас «Університет», який містить такі властивості:

- назва університету;

- адреса університету;
 - список факультетів, які входять до складу університету (використовуйте агрегацію);
 - ректор університету (використовуйте композицію).
2. Створіть клас «Факультет», який містить такі властивості:
- назва факультету;
 - кількість студентів;
 - декан факультету.
3. Створіть клас «Лектор», який містить такі властивості:
- прізвище та ім'я лектора;
 - посада лектора;
 - кафедра, на якій працює лектор (використовуйте композицію).
4. Створіть клас «Кафедра», який містить такі властивості:
- назва кафедри;
 - кількість викладачів;
 - голова кафедри.
5. Реалізуйте методи для додавання та видалення факультетів зі списку факультетів університету.
6. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Університет», «Факультет», «Лектор» та «Кафедра». Додайте факультети до списку факультетів університету та виведіть інформацію про них.
7. Продемонструйте успадкування, створивши похідний клас від «Лектор» (наприклад, «Доцент») з додатковими властивостями та методами.
8. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.
- Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 5

1. Створіть клас «Музичний гурт», який містить такі властивості:
- назва гурту;
 - стиль музики;
 - список музикантів, які входять до гурту (використовуйте агрегацію);
 - керівник гурту (використовуйте композицію).
2. Створіть клас «Музикант», який містить такі властивості:
- ім'я музиканта;
 - інструмент, на якому він грає.
3. Створіть клас «Інструмент», який містить такі властивості:
- назва інструмента;
 - рік виготовлення.
4. Реалізуйте методи для додавання та видалення музикантів зі списку музикантів гурту.

5. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Музичний гурт», «Музикант» та «Інструмент». Додайте музикантів до списку музикантів гурту та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від «Музикант» (наприклад, «Вокаліст») із додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 6

1. Створіть клас «Телефон», який містить такі властивості:

- марка телефона;
- модель телефона;
- екран (використовуйте композицію);
- камера (використовуйте композицію).

2. Створіть клас «Екран», який містить такі властивості:

- розмір екрана;
- тип екрана (lcd, oled тощо).

3. Створіть клас «Камера», який містить такі властивості:

- розмір матриці камери;
- роздільна здатність камери.

4. Реалізуйте методи для встановлення та отримання інформації про телефон, екран та камеру.

5. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Телефон», «Екран» та «Камера». Встановіть значення для властивостей та виведіть інформацію про телефон.

6. Продемонструйте успадкування, створивши похідний клас від «Телефон» (наприклад, «Смартфон») з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 7

1. Створіть клас «Університет», який містить такі властивості:

- назва університету;
- адреса університету;
- список факультетів, які належать до університету (використовуйте агрегацію).

2. Створіть клас «Факультет», який містить такі властивості:

- назва факультету;
- декан факультету (використовуйте композицію);

– список студентів, які навчаються на факультеті (використовуйте агрегацію).

1. Створіть клас «Декан», який містить такі властивості:

– ім'я декана;

– рік призначення на посаду декана.

5. Реалізуйте методи для додавання та видалення студентів зі списку студентів факультету.

6. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Університет», «Факультет» та «Декан». Додайте факультети до списку факультетів університету, додайте студентів до списку студентів факультету та виведіть інформацію про них.

7. Продемонструйте успадкування, створивши похідний клас від «Студент» (наприклад, «Магістр») з додатковими властивостями та методами.

8. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 8

1. Створіть клас «Магазин», який містить такі властивості:

– назва магазину;

– адреса магазину;

– список товарів, які є у магазині (використовуйте агрегацію).

2. Створіть клас «Товар», який містить такі властивості:

– назва товару;

– ціна товару.

3. Створіть клас «Кошик», який містить такі властивості:

– список товарів, які знаходяться у кошику (використовуйте композицію);

– загальна вартість товарів у кошику.

Реалізуйте методи для додавання та видалення товарів із кошика та розрахунку загальної вартості.

4. Напишіть демонстраційну функцію «main()», у якій створюються об'єкти класів «Магазин», «Товар» та «Кошик». Додайте товари до списку товарів магазину, додайте товари до кошика та виведіть інформацію про них та загальну вартість у кошику.

5. Продемонструйте успадкування, створивши похідний клас від «Товар» (наприклад, «Продукт») із додатковими властивостями та методами.

6. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 9

1. Створіть клас «Автомобіль», який містить такі властивості:

– марка автомобіля;

- модель автомобіля;
 - двигун (використовуйте композицію);
 - салон (використовуйте композицію).
2. Створіть клас «Двигун», який містить такі властивості:
- тип двигуна (бензиновий, дизельний тощо);
 - об'єм двигуна.
3. Створіть клас «Салон», який містить такі властивості:
- кількість місць в салоні;
 - тип салону (тканинний, шкіряний тощо).
4. Реалізуйте методи для встановлення та отримання інформації про автомобіль, двигун та салон.
5. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Автомобіль», «Двигун» та «Салон». Встановіть значення для властивостей та виведіть інформацію про автомобіль.
6. Продемонструйте успадкування, створивши похідний клас від класу «Автомобіль» (наприклад, «Спортивний автомобіль») із додатковими властивостями та методами.
7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.
- Не забудьте додати коментарі та пояснення до коду, щоб пояснити його функціональність та зрозумілість.

Варіант 10

1. Створіть клас «Банк», який містить такі властивості:
- назва банку;
 - адреса банку;
 - список відділень, які належать до банку (використовуйте агрегацію).
2. Створіть клас «Відділення», який містить такі властивості:
- назва відділення;
 - адреса відділення;
 - менеджер відділення (використовуйте композицію);
 - список клієнтів, які обслуговуються у відділенні (використовуйте агрегацію)
3. Створіть клас «Менеджер», який містить такі властивості:
- ім'я менеджера;
 - посада менеджера.
4. Реалізуйте методи для додавання та видалення клієнтів зі списку клієнтів відділення.
5. Напишіть демонстраційну функцію «main()», в якій створюються об'єкти класів «Банк», «Відділення» та «Менеджер». Додайте відділення до списку відділень банку, додайте клієнтів до списку клієнтів відділення та виведіть інформацію про них.
6. Продемонструйте успадкування, створивши похідний клас від класу «Клієнт» (наприклад, «VIP-Клієнт») із додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду.

Практична робота № 4

Поліморфізм. Обробка винятків

Мета: ознайомитись із поняттям поліморфізму у мові C# та навчитись використовувати віртуальні функції для досягнення поліморфізму. Також вивчити принципи обробки винятків у мові C#.

Теоретична частина

Поліморфізм – це принцип об’єктно-орієнтованого програмування, який дозволяє об’єктам одного класу використовуватися замість об’єктів інших класів, що мають спільний інтерфейс. Це означає, що об’єкти різних класів можуть бути оброблені і використані за допомогою загального коду без необхідності знати про конкретний клас об’єкта.

У C# поліморфізм можна реалізувати за допомогою віртуальних функцій.

Віртуальна функція – це функція, оголошена у базовому класі і перевизначена (або заміщена) у похідних класах. Ключове слово «virtual» перед оголошенням функції вказує, що ця функція є віртуальною.

Коли ви викликаєте віртуальну функцію на об’єкті, поведінка функції залежить від типу об’єкта, на якому вона викликається, а не від типу змінної або посилання, яке її містить. Це означає, що коли ви маєте посилання або вказівник на базовий клас, але викликаєте віртуальну функцію, то викликається реалізація цієї функції у відповідному похідному класі.

Оголошення віртуальної функції у базовому класі виглядає так:

```
using System;
```

```
class Base
```

```
{  
    // Віртуальний метод у базовому класі  
    public virtual void VirtualFunction()  
    {  
        // Реалізація функції у базовому класі  
        Console.WriteLine(«Реалізація у базовому класі»);  
    }  
}
```

```
class Derived : Base
```

```
{  
    // Переозначення (override) віртуального методу  
    public override void VirtualFunction()  
    {  
        Console.WriteLine(«Переозначена реалізація у похідному класі»);  
    }  
}
```

```
class Program
```

```
{  
    static void Main()  
    {  
        Base obj = new Derived();  
    }  
}
```

```

        obj.VirtualFunction(); // Викликається метод похідного класу завдяки поліморфізму
    }
}

```

Основна особливість віртуальних функцій полягає в тому, що під час виклику цієї функції на об'єктах через вказівники або посилання базового класу, виконується реалізація цієї функції у відповідному похідному класі, якщо така реалізація існує. Іншими словами, виклик віртуальної функції залежить від фактичного типу об'єкта, на якому вона викликається.

Приклад використання віртуальних функцій у похідних класах:

```
using System;
```

```

class Base
{
    public virtual void VirtualFunction()
    {
        Console.WriteLine(«Реалізація у базовому класі»);
    }
}

class Derived : Base
{
    public override void VirtualFunction()
    {
        Console.WriteLine(«Перевизначена реалізація функції у похідному класі»);
    }
}

class Program
{
    static void Main()
    {
        Base obj = new Derived();
        obj.VirtualFunction(); // Викликає перевизначений метод Derived
    }
}

```

У цьому прикладі ми створили похідний клас «Derived», який успадковує від базового класу «Base». У похідному класі ми перевизначили віртуальну функцію «virtualFunction()» зі зміною її реалізації.

Щоб позначити перевизначення функції у похідному класі, використовується ключове слово `override`. Це покращує зрозуміння коду та дозволяє компілятору перевірити, чи дійсно ми перевизначаємо функцію з базового класу.

У результаті ми можемо працювати з об'єктами похідних класів через вказівники або посилання базового класу і викликати їхні віртуальні функції. Водночас буде виконуватись відповідна реалізація цих функцій у відповідних похідних класах.

Приклад:

```
using System;

class Base
{
    public virtual void VirtualFunction()
    {
        Console.WriteLine(«Виклик функції з базового класу»);
    }
}

class Derived : Base
{
    public override void VirtualFunction()
    {
        Console.WriteLine(«Виклик функції з похідного класу»);
    }
}

class Program
{
    static void Main()
    {
        Base obj1 = new Derived();
        obj1.VirtualFunction(); // Виклик функції з похідного класу

        Base obj2 = new Base();
        obj2.VirtualFunction(); // Виклик функції з базового класу
    }
}
```

У цьому прикладі ми створюємо об'єкти через вказівники базового класу `Base`, але викликаємо їхні віртуальні функції. Перший виклик виконує функцію `virtualFunction()` з похідного класу «`Derived`», оскільки вказівник `obj1` посилається на об'єкт типу «`Derived`». Другий виклик виконує функцію `virtualFunction()` з базового класу «`Base`», оскільки вказівник `obj2` посилається на об'єкт типу «`Base`».

Таким чином, віртуальні функції дозволяють досягати поліморфізму в `C#`, що дозволяє з легкістю працювати з об'єктами різних класів через їх спільний інтерфейс.

Обробка винятків в мові `C#` здійснюється за допомогою механізму винятків (exceptions). Виняток – це спеціальний об'єкт, який викидається під час виникнення помилки або непередбаченої ситуації і може бути перехоплений і оброблений відповідним кодом.

Основні елементи обробки винятків у `C#`:

1. Викидання винятка. Для викидання винятка використовується оператор `throw`. Ви можете викинути будь-який об'єкт, який може бути скопійований або кинутий за допомогою оператора `throw`.

Приклад:

```
throw SomeException(); // Викидання об'єкта винятка
```

2. Перехоплення винятка. Для перехоплення винятка використовується блок `try-catch`. Ви можете вказати блок коду, в якому виняток може бути перехоплений, і надати відповідний обробник для обробки винятка.

Приклад:

```
try {  
    // Блок коду, в якому може виникнути виняток  
} catch (const SomeException& e) {  
    // Обробник для винятка типу SomeException  
} catch (const AnotherException& e) {  
    // Обробник для винятка типу AnotherException  
} catch (...) {  
    // Обробник для всіх інших винятків  
}
```

У блоці `try` вказується код, в якому виняток може виникнути. У блоках `catch` вказуються обробники для різних типів винятків. Обробники порівнюються з викинутими винятками на основі їхніх типів.

3. Розгортання стеку. Якщо виняток викидається в блоці `try`, система виконання розгортає стек (`stack unwinding`), що означає виклик деструкторів для об'єктів, які були створені в блоці `try`, перед тим, як виняток буде перехоплено.

4. Об'єкти винятків. Ви можете визначати власні типи винятків у вигляді класів. Ці класи можуть мати власні поля і методи, які дозволяють додатково ідентифікувати та обробляти винятки.

Приклад:

```
class MyException : public std::exception { public:  
    const char* what() const noexcept override {  
        return «Це мій виняток!»;  
    }  
};
```

У цьому прикладі `MyException` є власним класом винятка, який успадковує від `std::exception`. Він перевизначає метод `what()`, який повертає повідомлення про виняток.

5. Обробка винятків за межами функцій. Винятки можуть бути перехоплені не тільки у тому місці, де вони викидаються, але й у вищих рівнях виклику функцій. Якщо виняток не оброблений у поточній функції, він буде переданий наступному обробнику вищого рівня доти, доки виняток не буде перехоплено або програма не завершиться.

Це основні принципи обробки винятків у мові C#. Використання винятків дозволяє контролювати та обробляти помилки у програмі, дозволяючи зручно реагувати на непередбачені ситуації і забезпечувати безпеку виконання коду.

Завдання

Варіант 1

1. Розробіть систему керування транспортними засобами, яка міститиме різні типи транспортних засобів, такі як автомобілі, мотоцикли і вантажівки. Кожен з цих типів має власні характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен транспортний засіб повинен мати такі характеристики:

- запас палива (у літрах);
- швидкість (у км/год);
- вартість (у доларах).

Кожен тип транспортного засобу має власні додаткові характеристики:

1) автомобіль:

- кількість дверей;
- максимальна швидкість (у км/год);

2) мотоцикл:

- тип двигуна (бензиновий, електричний тощо);
- об'єм двигуна (у см³);

3) вантажівка:

- максимальне навантаження (у кг);
- кількість осей.

2. Створіть базовий абстрактний клас `Vehicle` з віртуальними функціями та використайте поліморфізм для реалізації методу обчислення вартості палива на відстань. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи `Car`, `Motorcycle` і `Truck`, які успадковуються від класу `Vehicle`. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо є негативний запас палива або швидкість, або якщо відстань для обчислення вартості палива менша або дорівнює нулю.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 2

1. Розробіть систему керування залізничним вокзалом, яка міститиме обробку різних типів поїздів, таких як пасажирські поїзди та вантажні поїзди. Кожен тип поїзда має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен поїзд має такі характеристики:

- номер потягу;
- пункт відправлення;
- пункт прибуття;
- час відправлення;

– час прибуття.

Кожен тип поїзда має власні додаткові характеристики:

1) пасажирський поїзд: кількість місць (сидячих та лежачих);

2) вантажний поїзд:

– максимальна вага вантажу (у тоннах);

– кількість вагонів.

2. Створіть базовий абстрактний клас `Train` із віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи `PassengerTrain` та `FreightTrain`, які успадковуються від класу `Train`. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо наявний час прибуття раніше або якщо він дорівнює часу відправлення.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 3

1. Розробіть систему керування продуктовим магазином, яка міститиме обробку різних типів продуктів, таких як фрукти, овочі та молочні продукти. Кожен тип продукту має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен продукт має такі характеристики:

– назва продукту;

– ціна (за одиницю товару);

– кількість на складі.

Кожен тип продукту має власні додаткові характеристики:

1) Фрукти:

– тип фрукту (яблуко, банан, апельсин тощо);

– вага (у кілограмах);

2) овочі:

– тип овочу (морква, капуста, буряк тощо);

– країна походження.

3) молочні продукти:

– тип продукту (молоко, йогурт, сир тощо);

– виробник.

2. Створіть базовий абстрактний клас `Product` із віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи `Fruit`, `Vegetable` та `DairyProduct`, які успадковуються від класу `Product`. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо наявна некоректна кількість на складі або негативна ціна.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 4

1. Розробіть систему керування банком, яка міститиме обробку різних типів банківських рахунків, таких як звичайний рахунок та рахунок із відсотковою ставкою. Кожен тип рахунку має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен банківський рахунок має наступні характеристики:

- номер рахунку;
- власник рахунку;
- баланс.

Кожен тип рахунку має власні додаткові характеристики:

- звичайний рахунок: мінімальний дозволений баланс;
- рахунок із відсотковою ставкою: відсоткова ставка.

2. Створіть базовий абстрактний клас BankAccount із віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи RegularAccount та InterestAccount, які успадковуються від класу BankAccount. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректного номера рахунку або негативної відсоткової ставки.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 5

1. Розробіть систему керування університетом, яка міститиме обробку різних типів студентів та викладачів. Кожен тип особи має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожна особа (студент або викладач) має такі характеристики:

- ім'я
- вік
- стать

Кожен тип особи має власні додаткові характеристики:

1) студент:

- курс;
- спеціальність;

2) викладач:

- посада;
- предмет, який він викладає.

2. Створіть базовий абстрактний клас Person із віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Student та Professor, які успадковуються від класу Person. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректного віку або порожнього імені.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 6

1. Розробіть систему керування магазином, яка міститиме обробку різних типів товарів. Кожен тип товару має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен товар має такі характеристики:

- назва;
- ціна;
- кількість на складі.

Кожен тип товару має власні додаткові характеристики:

1) меблі:

- матеріал;
- колір;

2) електроніка:

- бренд;
- гарантійний термін.

2. Створіть базовий абстрактний клас Product з віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Furniture та Electronics, які успадковуються від класу Product. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректної ціни товару або від'ємної кількості на складі.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 7

1. Розробіть систему керування човнами, яка міститиме обробку різних типів човнів. Кожен тип човна має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен човен має такі характеристики:

- назва;

- максимальна швидкість;
- вантажопідйомність.

Кожен тип човна має власні додаткові характеристики:

1) весловий човен:

- кількість весел;
- тип матеріалу;

2) моторний човен:

- потужність двигуна;
- тип палива.

2. Створіть базовий абстрактний клас Boat із віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Rowboat та Motorboat, які успадковуються від класу Boat. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректної швидкості човна або від'ємна вантажопідйомності.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 8

1. Розробіть систему керування одягом, яка міститиме різні типи одягу. Кожен тип одягу має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен одяг має такі характеристики:

- назва;
- розмір;
- колір.

Кожен тип одягу має власні додаткові характеристики:

1) футболка:

- матеріал;
- тип коміра;

2) штани:

- матеріал;
- тип застібки.

2. Створіть базовий абстрактний клас Clothing із віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи TShirt та Pants, які успадковуються від класу Clothing. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректного розміру одягу або порожнього імені.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 9

1. Розробіть систему управління аеропортом, яка міститиме різні типи повітряних суден. Кожен тип повітряного судна має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожне повітряне судно має такі характеристики:

- номер рейсу
- максимальна вага
- максимальна кількість пасажирів

Кожен тип повітряного судна має власні додаткові характеристики:

1) літак:

- модель;
- кількість двигунів;

2) гвинтокрил:

- кількість гвинтів;
- радіус дії.

2. Створіть базовий абстрактний клас Aircraft із віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Airplane та Helicopter, які успадковуються від класу Aircraft. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректного номера рейсу або від'ємної ваги повітряного судна.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 10

1. Розробіть систему управління лікарнею, яка міститиме різні типи медичних спеціалістів та пацієнтів. Кожен тип спеціаліста та пацієнта має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен медичний спеціаліст має такі характеристики:

- ім'я;
- спеціалізація;
- вік.

Кожен пацієнт має такі характеристики:

- ім'я;
- вік;
- діагноз.

Кожен тип медичного спеціаліста має власні додаткові характеристики:

1) лікар:

- стаж роботи;
- кількість успішно проведених операцій;
- 2) медсестра:
 - рівень кваліфікації;
 - кількість виконаних процедур.

2. Створіть базовий абстрактний клас `MedicalPersonnel` (медичний персонал) з віртуальними функціями та використайте поліморфізм для реалізації додаткових методів та функцій. Також додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи `Doctor` та `Nurse`, які успадковуються від класу `MedicalPersonnel`. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, у разі некоректного імені пацієнта або віку, який перевищує припустимий діапазон.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Практична робота № 5

Шаблони в C#

Мета: ознайомитись з основними поняттями шаблони та навчитись їх програмно реалізовувати мовою C++.

Теоретична частина

Шаблони є потужним інструментом у мові програмування C#. Вони дозволяють програмістам створювати загальні, параметризовані класи та функції, які можуть працювати з різними типами даних. Використання шаблонів дозволяє писати більш загальні й універсальні програми, що збільшує перевикористання коду, покращує його зрозумілість та підтримуваність.

Шаблони класів дозволяють визначити загальну структуру класу, незалежну від конкретного типу даних. Вони визначаються за допомогою ключового слова «template» і параметрів шаблону, які можуть бути типами даних, константами або іншими шаблонами. Наприклад, ось простий шаблон класу «Stack», який реалізує структуру стеку:

```
using System;
using System.Collections.Generic;

class Stack<T>
{
    private List<T> elements = new List<T>(); // Зберігає елементи стеку

    public void Push(T element)
    {
        elements.Add(element);
    }

    public void Pop()
    {
        if (!Empty())
        {
            elements.RemoveAt(elements.Count - 1);
        }
        else
        {
            throw new InvalidOperationException(«Стек порожній»);
        }
    }

    public T Top()
    {
        if (!Empty())
        {
            return elements[elements.Count - 1];
        }
        else
    }
```

```

        {
            throw new InvalidOperationException(«Стек порожній»);
        }
    }

    public bool Empty()
    {
        return elements.Count == 0;
    }
}

```

У цьому прикладі тип даних «Т» є параметром шаблону. Клас «Stack» може бути використаний для створення стеку будь-якого типу даних.

Приклад:

```
class Program
```

```

{
    static void Main()
    {
        Stack<int> stack = new Stack<int>();

        stack.Push(10);
        stack.Push(20);
        Console.WriteLine(«Top: « + stack.Top()); // Виведе: Top: 20
        stack.Pop();
        Console.WriteLine(«Top: « + stack.Top()); // Виведе: Top: 10
    }
}

```

Крім шаблонів класів, у C# також є шаблони функцій. Вони дозволяють визначати функції, які працюють із різними типами даних. Шаблони функцій використовують таку саму синтаксичну конструкцію з ключовим словом «template» і параметрами шаблону. Наприклад, ось проста шаблонна функція «max», яка повертає більший із двох переданих значень:

```

using System;

class Program
{
    static T Max<T>(T a, T b) where T : IComparable<T>
    {
        return a.CompareTo(b) > 0 ? a : b;
    }

    static void Main()
    {
        Console.WriteLine(Max(5, 10)); // Виведе: 10
        Console.WriteLine(Max(«apple», «zebra»)); // Виведе: zebra
    }
}

```

Цю функцію можна використати для порівняння значень будь-якого типу даних, для якого визначені оператори порівняння.

```
using System;
```

```
class Program
{
    static T Max<T>(T a, T b) where T : IComparable<T>
    {
        return a.CompareTo(b) > 0 ? a : b;
    }

    static void Main()
    {
        int result1 = Max(10, 20);    // Поверне 20
        double result2 = Max(3.14, 2.71); // Поверне 3.14

        Console.WriteLine(«Max int: « + result1);
        Console.WriteLine(«Max double: « + result2);
    }
}
```

Шаблони в C# також підтримують спеціалізацію, що дозволяє визначати варіанти шаблону для конкретних типів даних або спеціальних випадків. Зазвичай спеціалізацію використовують для оптимізації або визначення специфічної поведінки для певного типу даних.

Шаблони в C# – це потужний інструмент, який дозволяє створювати загальні та універсальні програми. Вони забезпечують перевикористання коду, покращують зрозумілість і підтримуваність програм, а також дають можливість програмістам працювати з різними типами даних без необхідності дублювання коду.

Завдання

Варіант 1

Створіть програму для керування бібліотекою книжок. Кожна книжка має заголовок, автора та рік видання. Необхідно створити клас Book, що містить ці дані.

Крім того, необхідно створити шаблонний клас Library, який містить колекцію книжок. Цей клас має методи для додавання книжки, видалення книжки за заголовком, виведення списку всіх книжок та пошуку книжок за автором.

Забезпечте універсальність класу Library, використовуючи шаблонний параметр для типу книжки. Це дозволить працювати з різними типами книжок.

Для демонстрації роботи програми:

1. Створіть об'єкт Library для зберігання книжок.
2. Додайте до бібліотеки декілька книжок різних типів, наприклад, художніх, наукових, документальних тощо.
3. Виведіть на екран список всіх книжок у бібліотеці.
4. Знайди та виведи на екран усі книжки певного автора.

5. Видаліть із бібліотеки книжку за заголовком.
6. Знову виведіть на екран оновлений список книжок у бібліотеці.

Не забудьте використати шаблонний клас `Library` для зберігання книжок будь-якого типу. Також подбайте про те, щоб клас `Book` мав методи доступу до своїх полів (заголовку, автора, року видання).

Варіант 2

Створіть програму для керування студентським списком. Кожен студент має ім'я, прізвище та середній бал. Необхідно створити клас `Student`, який містить ці дані.

Крім того, необхідно створити шаблонний клас `StudentList`, який містить колекцію студентів. Цей клас має методи для додавання студента, видалення студента за прізвищем, виведення списку всіх студентів та пошуку студентів за середнім балом.

Забезпечте універсальність класу `StudentList`, використовуючи шаблонний параметр для типу студента. Це дозволить працювати з різними типами студентів (наприклад, з інженерного факультету, медичного факультету тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт `StudentList` для зберігання студентів.
2. Додайте до списку декілька студентів із різних факультетів.
3. Виведіть на екран список всіх студентів.
4. Знайдіть та виведіть на екран усіх студентів із певним середнім балом.
5. Видаліть зі списку студента за прізвищем.
6. Знову виведіть на екран оновлений список студентів у списку.

Не забудьте використати шаблонний клас `StudentList` для зберігання студентів будь-якого типу. Також подбайте про те, щоб клас `Student` мав методи доступу до своїх полів (імені, прізвища, середнього бала).

Варіант 3

Створіть програму для керування банківським рахунком клієнтів. Кожен клієнт має ім'я, прізвище та баланс на рахунку. Необхідно створити клас `Customer`, який містить ці дані.

Крім того, необхідно створити шаблонний клас `BankAccount`, який містить банківський рахунок. Цей клас має методи для додавання коштів на рахунок, зняття коштів з рахунку, переказу коштів між рахунками та виведення балансу рахунку.

Забезпечте універсальність класу `BankAccount`, використовуючи шаблонний параметр для типу клієнта. Це дозволить працювати з різними типами клієнтів (наприклад, фізичними особами, юридичними особами тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт `BankAccount` для керування рахунком клієнта.
2. Додайте кошти на рахунок за допомогою методу `deposit()`.
3. Зніміть кошти з рахунку за допомогою методу `withdraw()`.

4. Перекажіть кошти з одного рахунку на інший за допомогою методу `transfer()`.

5. Виведіть на екран баланс рахунку за допомогою методу `getBalance()`.

Не забудьте використати шаблонний клас `BankAccount` для керування рахунками різних типів клієнтів. Також подбайте про те, щоб клас `Customer` мав методи доступу до своїх полів (імені, прізвища, балансу).

Варіант 4

Створіть програму для керування складом товарів у магазині. Кожен товар має назву, ціну та кількість на складі. Необхідно створити клас `Product`, який містить ці дані.

Крім того, потрібно створити шаблонний клас `Inventory`, який містить інвентар магазину. Цей клас має методи для додавання товару на склад, видалення товару зі складу за назвою, виведення списку всіх товарів та пошуку товару за ціною.

Забезпечте універсальність класу `Inventory`, використовуючи шаблонний параметр для типу товару. Це дозволить працювати з різними типами товарів (наприклад, продуктами харчування, електронікою, одягом тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт `Inventory` для зберігання товарів на складі.
2. Додайте до інвентаря декілька товарів різних типів.
3. Виведіть на екран список всіх товарів на складі.
4. Знайдіть та виведіть на екран всі товари за певною ціною.
5. Видаліть з інвентаря товар за назвою.
6. Знову виведіть на екран оновлений список товарів на складі.

Не забудьте використати шаблонний клас `Inventory` для зберігання товарів будь-якого типу. Також подбайте про те, щоб клас `Product` мав методи доступу до своїх полів (назви, ціни, кількості).

Варіант 5

Створіть програму для керування рестораном. У ресторані є різні страви, які клієнти можуть замовити. Кожна страву має назву, опис та ціну. Необхідно створити клас `Dish`, який містить ці дані.

Крім того, потрібно створити шаблонний клас `Menu`, який містить меню ресторану. Цей клас має методи для додавання страви до меню, видалення страви за назвою, виведення списку всіх страв та пошуку страви за ціною.

Забезпечте універсальність класу `Menu`, використовуючи шаблонний параметр для типу страви. Це дозволить працювати з різними типами страв (наприклад, стравами з м'ясом, стравами для вегетаріанців тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт `Menu` для зберігання страв.
2. Додайте до меню декілька різних страв.
3. Виведіть на екран список всіх страв у меню.
4. Знайдіть та виведіть на екран всі страви за певною ціною.

5. Видаліть з меню страву за назвою.
6. Знову виведіть на екран оновлений список страв у меню.

Не забудьте використати шаблонний клас Menu для зберігання страв будь-якого типу. Також подбайте про те, щоб клас Dish мав методи доступу до своїх полів (назви, опису, ціни).

Варіант 6

Створіть програму для керування туристичним агентством. Туристичне агентство пропонує різні тури та послуги для подорожей. Кожен тур має назву, опис, ціну та тривалість. Необхідно створити клас Tour, який містить ці дані.

Крім того, потрібно створити шаблонний клас TravelAgency, який уособлює туристичне агентство. Цей клас має методи для додавання туру до агентства, видалення туру за назвою, виведення списку всіх доступних турів та пошуку туру за ціною.

Забезпечте універсальність класу TravelAgency, використовуючи шаблонний параметр для типу туру. Це дозволить працювати з різними типами турів (наприклад, пляжними турами, екскурсійними турами, гірськолижними турами тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт TravelAgency для зберігання турів.
2. Додайте до агентства декілька різних турів.
3. Виведіть на екран список всіх доступних турів.
4. Знайдіть та виведіть на екран всі тури за певною ціною.
5. Видаліть з агентства тур за назвою.
6. Знову виведіть на екран оновлений список доступних турів.

Не забудьте використати шаблонний клас TravelAgency для зберігання турів будь-якого типу. Також подбайте про те, щоб клас Tour мав методи доступу до своїх полів (назви, опису, ціни, тривалості).

Варіант 7

Створіть програму для керування автомобільним сервісним центром. Сервісний центр займається обслуговуванням автомобілів та наданням різних послуг. Кожен автомобіль має марку, модель та рік випуску. Необхідно створити клас Car, який містить ці дані.

Крім того, потрібно створити шаблонний клас ServiceCenter, який уособлює сервісний центр. Цей клас має методи для додавання автомобіля до списку обслуговування, видалення автомобіля за маркою, виведення списку всіх автомобілів у сервісному центрі та пошуку автомобіля за роком випуску.

Забезпечте універсальність класу ServiceCenter, використовуючи шаблонний параметр для типу автомобіля. Це дозволить працювати з різними типами автомобілів (наприклад, легковими, вантажними, спортивними тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт ServiceCenter для зберігання автомобілів.
2. Додайте до сервісного центру декілька різних автомобілів.
3. Виведіть на екран список всіх автомобілів у сервісному центрі.

4. Знайдіть та виведіть на екран всі автомобілі за певним роком випуску.
5. Видаліть із сервісного центру автомобіль за маркою.
6. Знову виведіть на екран оновлений список автомобілів у сервісному центрі. Не забудьте використати шаблонний клас ServiceCenter для зберігання автомобілів будь-якого типу. Також подбайте про те, щоб клас Car мав методи доступу до своїх полів (марки, моделі, року випуску).

Варіант 8

Створіть програму для керування музичним магазином. Музичний магазин продає різні музичні інструменти та аксесуари. Кожен інструмент має назву, тип та ціну. Необхідно створити клас Instrument, який містить ці дані.

Крім того, потрібно створити шаблонний клас MusicStore, який уособлює музичний магазин. Цей клас має методи для додавання інструмента до асортименту магазину, видалення інструмента за назвою, виведення списку всіх доступних інструментів та пошуку інструмента за типом.

Забезпечте універсальність класу MusicStore, використовуючи шаблонний параметр для типу інструмента. Це дозволить працювати з різними типами інструментів (наприклад, гітарами, барабанами, клавішними тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт MusicStore для зберігання інструментів.
2. Додайте до магазину декілька різних інструментів.
3. Виведіть на екран список всіх доступних інструментів у магазині.
4. Знайдіть та виведіть на екран всі інструменти певного типу.
5. Видаліть з магазину інструмент за назвою.
6. Знову виведіть на екран оновлений список доступних інструментів у магазині.

Не забудьте використати шаблонний клас MusicStore для зберігання інструментів будь-якого типу. Також подбайте про те, щоб клас Instrument мав методи доступу до своїх полів (назви, типу, ціни).

Варіант 9

Створіть програму для керування спортивним клубом. Спортивний клуб має різні секції, кожна з яких спеціалізується на певному виді спорту. Кожна секція має назву, тренера та кількість учасників. Необхідно створити клас Section, який містить ці дані.

Крім того, потрібно створити шаблонний клас SportsClub, який містить спортивний клуб. Цей клас має методи для додавання секції до клубу, видалення секції за назвою, виведення списку всіх доступних секцій та пошуку секції за кількістю учасників.

Забезпечте універсальність класу SportsClub, використовуючи шаблонний параметр для типу секції. Це дозволить працювати з різними видами спорту (наприклад, футболом, баскетболом, плаванням тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт SportsClub для зберігання секцій.

2. Додайте до клубу декілька різних секцій.
3. Виведіть на екран список всіх доступних секцій у клубі.
4. Знайдіть та виведіть на екран всі секції з певною кількістю учасників.
5. Видаліть з клубу секцію за назвою.
6. Знову виведіть на екран оновлений список доступних секцій у клубі.

Не забудьте використати шаблонний клас `SportsClub` для зберігання секцій будь-якого типу. Також подбайте про те, щоб клас `Section` мав методи доступу до своїх полів (назви, тренера, кількості учасників).

Варіант 10

Створіть програму для керування онлайн-курсами. Кожен курс має назву, викладача та кількість студентів. Необхідно створити клас `Course`, який містить ці дані.

Крім того, потрібно створити шаблонний клас `OnlineLearningPlatform`, який містить платформу для онлайн-навчання. Цей клас має методи для додавання курсу до платформи, видалення курсу за назвою, виведення списку всіх доступних курсів та пошуку курсу за кількістю студентів.

Забезпечте універсальність класу `OnlineLearningPlatform`, використовуючи шаблонний параметр для типу курсу. Це дозволить працювати з різними типами курсів (наприклад, програмуванням, мовами, маркетингом тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт `OnlineLearningPlatform` для зберігання курсів.
2. Додайте до платформи декілька різних курсів.
3. Виведіть на екран список всіх доступних курсів на платформі.
4. Знайдіть та виведіть на екран всі курси з певною кількістю студентів.
5. Видаліть із платформи курс за назвою.
6. Знову виведіть на екран оновлений список доступних курсів на платформі.

Не забудьте використати шаблонний клас `OnlineLearningPlatform` для зберігання курсів будь-якого типу. Також подбайте про те, щоб клас `Course` мав методи доступу до своїх полів (назви, викладача, кількості студентів).

Практична робота № 6

Контейнерні класи. Стандартна бібліотека шаблонів у C#

Мета: ознайомитись із контейнерними класами та навчитись їх програмно реалізовувати мовою C#.

Теоретична частина

Контейнерний клас (або «клас-контейнер») у мові C# – це клас, призначений для зберігання і організації декількох об'єктів певного типу даних (користувацьких чи фундаментальних).

У мові програмування C# контейнерні класи є класами, які призначені для зберігання та керування колекціями об'єктів. Вони надають зручний інтерфейс

для додавання, видалення і доступу до елементів у колекції.

Контейнерні класи в C# доступні завдяки бібліотеці стандартного шаблону (STL – Standard Template Library), яка містить різні типи контейнерів, такі як вектори, списки, множини, асоціативні масиви тощо.

Основні типи контейнерів, які можна знайти у STL, містять такі:

1. Вектор (vector): послідовний динамічний масив елементів.
2. Список (list): двоспрямований зв'язаний список елементів.
3. Двійкове дерево (binary tree): структура даних, де кожен вузол може мати не більше двох нащадків.
4. Мапа (map): асоціативний масив, що зберігає пари «ключ – значення».
5. Множина (set): колекція унікальних елементів без поняття порядку.

Це лише кілька прикладів контейнерів, що надаються STL. Кожен контейнер має свої особливості й методи для додавання, видалення, пошуку та інших операцій, що можуть бути виконані з елементами колекції.

Використання контейнерних класів дозволяє забезпечити ефективне управління даними та спростити роботу з колекціями у мові програмування C#.

Наприклад використання контейнерного класу vector із бібліотеки STL у C#:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Створення об'єкта списку для зберігання цілих чисел
        List<int> numbers = new List<int>();

        // Додавання елементів до списку
        numbers.Add(10);
        numbers.Add(20);
    }
}
```

```

numbers.Add(30);

// Виведення елементів списку
Console.Write(«Елементи списку: »);
for (int i = 0; i < numbers.Count; i++)
{
    Console.Write(numbers[i] + « »);
}
Console.WriteLine();

// Видалення останнього елемента зі списку
numbers.RemoveAt(numbers.Count - 1);

// Оновлення значення елемента
numbers[1] = 50;

// Виведення оновлених елементів списку
Console.Write(«Оновлені елементи списку: »);
foreach (int number in numbers)
{
    Console.Write(number + « »);
}
Console.WriteLine();

// Отримання розміру списку
Console.WriteLine(«Розмір списку: « + numbers.Count);
}
}

```

У цьому прикладі спочатку створюється об'єкт вектора `numbers`, який зберігатиме цілі числа. За допомогою функції `push_back()` додаються числа 10, 20 і 30 до вектора. Потім елемент 30 видаляється за допомогою функції `pop_back()`, а елемент з індексом 1 (20) оновлюється на 50. Нарешті, елементи вектора виводяться на екран за допомогою циклу, а розмір вектора виводиться за допомогою функції `size()`.

Вивід програми буде таким:

```

Елементи вектора: 10 20 30
Оновлені елементи вектора: 10 50
Розмір вектора: 2

```

Завдання

Варіант 1

Розробіть систему керування магазином електроніки. У цій системі необхідно зберігати і керувати інформацією про продукти, які доступні у магазині. Для цього вам потрібно використовувати контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком продуктів:

1. Додайте декілька продуктів до інвентаря, включно з їхньою назвою, ціною та кількістю на складі.
2. Виведіть список продуктів у інвентарі.
3. Оновіть кількість одного або декількох продуктів у списку.
4. Виведіть оновлений список продуктів у інвентарі.
5. Обчисліть загальну вартість усього інвентаря та виведіть її.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 2

Розробіть систему управління бібліотекою. У цій системі потрібно зберігати та керувати інформацією про книги, що знаходяться у бібліотеці.

Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком книг:

1. Додайте декілька книг до бібліотеки, включно з їхньою назвою, автором, роком видання та ISBN.
2. Виведіть список книг у бібліотеці.
3. Видаліть одну або декілька книг зі списку за ISBN.
4. Проведіть пошук книг за автором і виведіть результат.
5. Проведіть пошук книг за роком видання і виведіть результат.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 3

Розробіть систему управління завданнями. У цій системі потрібно зберігати та керувати списком завдань. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком завдань.

1. Додайте декілька завдань до списку, включно з їхньою назвою, описом та дедлайном.
2. Виведіть список завдань.
3. Видаліть одне або декілька завдань зі списку за назвою.
4. Виведіть оновлений список завдань.
5. Виведіть список завдань із певним дедлайном.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 4

Розробіть систему управління складом магазину автозапчастин. У цій системі потрібно зберігати та керувати інформацією про наявні автозапчастини. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком запчастин:

1. Додайте декілька запчастин до складу, включно з їхньою назвою, виробником, ціною та кількістю на складі.
2. Виведіть список запчастин на складі.
3. Видаліть одну або декілька запчастин зі складу за назвою.

4. Оновіть ціну та кількість для певної запчастини.
5. Виведіть оновлений список запчастин на складі.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 5

Розробіть систему управління студентськими курсами. У цій системі потрібно зберігати та керувати інформацією про курси та студентів, які до них записалися. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними. Використайте знання, набуті на минулих заняттях, для побудови програмного продукту:

1. Додайте декілька курсів до списку, включно з їхньою назвою, викладачем та кількістю годин.
2. Виведіть список курсів та кількості записаних студентів на кожний курс.
3. Видаліть один або декілька курсів зі списку за назвою.
4. Запишіть декілька студентів на певні курси.
5. Виведіть оновлений список курсів та кількості записаних студентів на кожний курс.
6. Виведіть список студентів, записаних на певний курс.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 6

Розробіть систему керування запасами продуктів у супермаркеті. У цій системі потрібно зберігати та керувати інформацією про наявні продукти, їх кількість та ціни. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- 3) Додайте декілька продуктів до списку, включно з їхньою назвою, категорією, ціною та кількістю на складі.
- 4) Виведіть повний список продуктів на складі.
- 5) Видаліть один або декілька продуктів зі списку за назвою.
- 6) Оновіть ціну та кількість для певного продукту.
- 7) Виведіть оновлений список продуктів на складі.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 7

Розробіть систему керування готелями. У цій системі потрібно зберігати та керувати інформацією про доступні номери, резервації та клієнтів.

Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними:

1. Додайте декілька кімнат до списку доступних кімнат, включно з їхнім номером, категорією, ціною та доступністю.

2. Виведіть список доступних кімнат.
3. Здійсніть резервацію для декількох кімнат, включно з номером кімнати, датою початку та закінчення резервації та кількість гостей.
4. Виведіть список всіх резервацій.
5. Скасуйте резервацію для певної кімнати за номером.
6. Виведіть оновлений список доступних кімнат.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 8

Розробіть систему керування поштовими відправленнями. У цій системі потрібно зберігати та керувати інформацією про відправлення, включно з даними про відправника, одержувача, вагу та тип поштового відправлення. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними:

1. Додайте декілька відправлень до списку, включно з даними про відправника, одержувача, вагу та тип відправлення.
2. Виведіть список всіх відправлень.
3. Видаліть певне відправлення за номером.
4. Оновіть вагу та тип для певного відправлення.
5. Виведіть оновлений список всіх відправлень.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 9

Розробіть систему керування студентськими оцінками. У цій системі потрібно зберігати та керувати оцінками студентів за різними предметами. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними:

1. Додайте декілька студентів до списку, включно з їхніми іменем, прізвищем та початковими оцінками.
2. Виведіть список оцінок всіх студентів.
3. Видаліть певного студента за ім'ям та прізвищем.
4. Додайте оцінку певному студенту.
5. Видаліть оцінку у певного студента.
6. Виведіть оновлений список оцінок всіх студентів.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 10

Розробіть систему керування замовленнями у ресторані. У цій системі потрібно зберігати та обробляти дані про замовлення страв, включно з назвами

страв, їхньою кількістю та ціною. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними:

1. Додайте декілька страв до замовлення, вказавши їх назви та ціни.
2. Виведіть загальну суму замовлення.
3. Видаліть певну страву зі замовлення за назвою.
4. Додайте інші страви до замовлення.
5. Виведіть оновлену загальну суму замовлення.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Об'єктно-орієнтоване програмування [Електрон. ресурс] : метод. посіб. до лек. та лаб. занять для студентів ф-ту математики, фізики та інформ. технол. першого (бакалавр.) рівня освіти спец. 122 – Комп'ютерні науки / уклад. : А. Л. Рачинська, О. А. Недєва, К. С. Палій. – Електрон. текст. дані. – Одеса : Одес. нац. ун-т ім. І. І. Мечникова, 2024. – 338 с. – Режим доступу: <https://dspace.onu.edu.ua/bitstreams/68babdee-36dd-42e9-aeaa-ecb0a23abd2b/download>, вільний (дата звернення: 29.03.2025). – Назва з екрана.
2. Посібник із класів і об'єктів C# із прикладами [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу: <https://www.guru99.com/uk/c-sharp-class-object.html>, вільний (дата звернення: 29.03.2025). – Назва з екрана.
3. NET 6 books, will update.net 7 version [Electronic resource]. – Electronic text data. – Regime of access: <https://github.com/pty819/csharpbooks/tree/main>, free (date of the application: 02.03.2025). – Header from the screen.
4. Tutorial: Get started with ASP.NET Core [Electronic resource]. – Electronic text data. – Regime of access: <https://learn.microsoft.com/en-us/aspnet/core/getting-started/?view=aspnetcore-9.0>, free (date of the application: 14.05.2025). – Header from the screen.
5. Start building with these Microsoft products [Electronic resource]. – Electronic text data. – Regime of access: <https://developer.microsoft.com/en-us/>, free (date of the application: 12.01.2025). – Header from the screen.
6. Desktop Guide for .NET and .NET Framework [Electronic resource]. – Electronic text data. – Regime of access: <https://learn.microsoft.com/uk-ua/dotnet/desktop/>, free (date of the application: 03.03.2025). – Header from the screen.
7. Resources for Developers, by Developers [Electronic resource]. – Electronic text data. – Regime of access: <https://developer.mozilla.org/en-US/>, вільний (дата звернення: 03.04.2025). – Назва з екрана.
8. Томка Ю. Я. Об'єктно-орієнтоване програмування мовою C# [Електрон. ресурс] / Ю. Я. Томка. – Чернівці : Технодрук, 2022. – 504 с. –

Електрон. текст. дані. – Режим доступу :
https://archer.chnu.edu.ua/xmlui/bitstream/handle/123456789/6704/Tomka_OOP_2022.pdf?sequence=1&isAllowed=y, вільний (дата звернення: 15.04.2025). – Назва з екрана.

9. Об'єктно-орієнтоване програмування. Лабораторний практикум [Електрон. ресурс] : навч. посіб. для студентів спеціальності 113 – «Прикладна математика» / КПІ ім. Ігоря Сікорського ; уклад. : Н. М. Куссуль, А. Ю. Шелестов, А. М. Лавренюк, Д. В. Тітков. – Електрон. текст. дані. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 56 с. – Режим доступу: <https://ela.kpi.ua/handle/123456789/43489>, вільний (дата звернення: 15.04.2025). – Назва з екрана.

10. Омельчук Л. Л. Об'єктно-орієнтоване програмування. Лабораторний практикум [Електрон. ресурс] : навч. посіб. / Л. Л. Омельчук. – Київ : 2021. – 265 с. – Електрон. текст. дані. – Режим доступу: http://csc.knu.ua/media/filer_public/29/22/29222da5-a099-4c6e-8b21-87faa00cb1bd/oop_2021_2022_laboratornii_praktikum.pdf, вільний (дата звернення: 19.04.2025). – Назва з екрана.

11. Курс «Об'єктно-орієнтоване програмування» у віртуальному освітньому середовищі на платформі MOODLE ХНУМГ ім. О. М. Бекетова. [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу: <https://dl.kname.edu.ua/course/view.php?id=801>, вільний (дата звернення: 20.04.2025). – Назва з екрана.

12. Онлайн-курси Prometheus [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://prometheus.org.ua/>, вільний (дата звернення: 15.04.2025). – Назва з екрана.

13. Онлайн-курси Coursera. [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://www.coursera.org>, вільний (дата звернення: 15.04.2024). – Назва з екрана.

14. Академія Cisco [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://www.netacad.com>, вільний (дата звернення: 19.04.2025). – Назва з екрана.

15. Онлайн ресурс з інформаційних технологій [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://habr.com>, вільний (дата звернення: 19.04.2025). – Назва з екрана.

16. Онлайн-ресурс із інфо [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://dou.ua/>, вільний (дата звернення: 19.04.2025). – Назва з екрана.

17. Пошукова система [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://www.google.com/>, вільний (дата звернення: 21.04.2025). – Назва з екрана.

18. Онлайн-ресурс перегляду відеоуроків [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://www.youtube.com>, вільний (дата звернення: 21.04.2025). – Назва з екрана.

19. Онлайн-ресурс aCode. [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Оновлюється постійно. – Режим доступу: <https://acode.com.ua/>, вільний (дата звернення: 21.04.2025). – Назва з екрана.

20. C# Reference, сайт розробників MSDN [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу: <https://code.visualstudio.com/docs/languages/csharp>, вільний (дата звернення: 27.04.2025). – Назва з екрана.

Електронне навчальне видання

Методичні рекомендації
до виконання практичних та самостійних робіт
із навчальної дисципліни

«ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

*(для здобувачів першого (бакалаврського) рівня вищої освіти денної форми
навчання зі спеціальностей F3, 122 – Комп’ютерні науки)*

Укладач **БРЕДІХІН** Володимир Михайлович

Відповідальний за випуск *М. В. Булаєнко*
Редактор *Б. О. Хільська*
Комп’ютерне верстання *В. М. Бредіхін*

План 2024, поз. 284М

Підп. до друку 03.11.2025. Формат 60 × 84/16.
Ум. друк. арк. 3,2.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Чорноглазівська, 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб’єкта видавничої справи:
ДК № 8386 від 14.07.2025.