Кам’янець-Подільський національний університет імені Івана Огієнка

КАФЕДРА КОМП’ЮТЕРНИХ НАУК

Навчальна дисципліна «Об’єктно-орієнтоване програмування»

ЛАБОРАТОРНА РОБОТА #0105

Тема:

Використання інтерфейсів

Варіант №3

**Виконала**:  
студентка 1-го курсу  
групи KNms1-B24  
Некрасова Ю.Д.

**Прийняв**:  
Слободянюк О.В.

Кам’янець-Подільський – 2025

1. **Короткі теоретичні відомості**

**Визначення та Застосування інтерфейсів**. Інтерфейси — це набір абстрактних методів і властивостей, які клас зобов’язується реалізувати. Вони визначають контракт без реалізації, що забезпечує можливість поліморфізму та створення кодів із чітким розподілом обов’язків. Інтерфейси застосовуються для забезпечення взаємодії між несумісними класами, створенням API та розширюваності систем.

**Явна реалізація інтерфейсів. Реалізація інтерфейсів у базових та похідних класах**. Явна реалізація інтерфейсів дозволяє реалізувати методи інтерфейсу так, що вони доступні лише через посилання на цей інтерфейс. Це корисно для уникнення конфліктів імен або коли реалізація повинна бути прихованою від загального використання. Базові та похідні класи можуть реалізовувати інтерфейси, причому похідні класи можуть перевизначати або розширювати реалізації базового класу.

**Наслідування інтерфейсів**. Інтерфейси можуть успадковувати інші інтерфейси, що дозволяє створювати складніші структури на основі простіших контрактів. Це забезпечує розширення функціональності без порушення принципу єдиного обов’язку та підтримує концепцію композиції через множинне наслідування, доступне для інтерфейсів.

**Інтерфейси в узагальненнях**. Узагальнення (Generics) дозволяють створювати класи, методи та інтерфейси з параметрами типів. Інтерфейси можуть використовуватися як обмеження для узагальнених типів, що дозволяє створювати універсальні алгоритми з підтримкою конкретних інтерфейсів. Наприклад, можна обмежити типи даних для сортування до тих, що реалізують IComparable.

**Копіювання об'єктів. Інтерфейс ICloneable. Сортування об'єктів. Інтерфейс IComparable**.

* ICloneable визначає метод Clone(), який створює копію об’єкта. Він може реалізовувати як поверхневе (shallow), так і глибоке (deep) копіювання.
* IComparable визначає метод CompareTo(), що дозволяє об’єктам порівнювати себе з іншими об’єктами того ж типу. Це корисно для сортування та пошуку елементів у колекціях.

**Коваріантність та контраваріантність узагальнених інтерфейсів**. Коваріантність дозволяє використовувати більш конкретний тип, ніж вказаний узагальнений тип (переважно для вихідних значень, як у делегатах або колекціях лише для читання). Контраваріантність дозволяє використовувати більш загальний тип (переважно для вхідних значень, як у колекціях лише для запису або делегатах з параметрами). Це дозволяє забезпечити більшу гнучкість та сумісність при роботі з узагальненими типами.

1. **Умови завдання**

Задача 1. Інтерфейс і клас.

Створіть інтерфейс ILoginable з методом Login(). Реалізуйте інтерфейс у класі User. Створіть об’єкт класу і викличте метод через інтерфейс.

1. **Послідовність виконання роботи**

Інтерфейс ILoginable містить оголошення методу Login(), який обов’язково потрібно реалізувати в класі.

Клас User реалізує інтерфейс, додаючи метод Login() з конкретною логікою.

Виклик методу через інтерфейс:

* Об'єкт user приводиться до типу ILoginable (loginableUser).
* Метод Login() викликається через інтерфейсне посилання.

1. **Код програми**

using System;

namespace InterfaceExample

{

// Оголошення інтерфейсу з методом Login()

public interface ILoginable

{

void Login();

}

// Клас User, що реалізує інтерфейс ILoginable

public class User : ILoginable

{

public string Username { get; set; }

public User(string username)

{

Username = username;

}

// Реалізація методу Login()

public void Login()

{

Console.WriteLine($"{Username} has successfully logged in.");

}

}

class Program

{

static void Main(string[] args)

{

// Створення об’єкта класу User

User user = new User("Andrew");

// Виклик методу Login через інтерфейс

ILoginable loginableUser = user; // Приведення об’єкта User до типу ILoginable

loginableUser.Login();

// Або можна викликати метод безпосередньо

user.Login();

}

}

}

1. **Приклад виконання програми**

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст, створений ШІ, може бути неправильним.

1. **Умови завдання**

Задача 2. Явна реалізація в контексті.

Створіть інтерфейс ISaveable з методом Save(). Реалізуйте інтерфейс в класі File явно, щоб метод був доступний тільки через інтерфейс.

1. **Послідовність виконання роботи**

Інтерфейс ISaveable містить оголошення методу Save(), який обов’язково потрібно реалізувати в класі.

Клас File реалізує інтерфейс ISaveable явно, додаючи метод Save() з логікою збереження файлу. Явна реалізація робить цей метод доступним лише через інтерфейсне посилання.

Виклик методу через інтерфейс:

* Об'єкт myFile приводиться до типу ISaveable (saveableFile).
* Метод Save() викликається через інтерфейсне посилання saveableFile.Save().

Прямий виклик myFile.Save() неможливий, оскільки метод Save() реалізований явно.

1. **Код програми**

using System;

namespace InterfaceExample

{

// Інтерфейс з методом Save()

public interface ISaveable

{

void Save();

}

// Клас File, що явно реалізує інтерфейс ISaveable

public class File : ISaveable

{

public string FileName { get; set; }

public File(string fileName)

{

FileName = fileName;

}

// Явна реалізація методу Save()

void ISaveable.Save()

{

Console.WriteLine($"File '{FileName}' has been successfully saved.");

}

}

class Program

{

static void Main(string[] args)

{

// Створення об’єкта класу File

File myFile = new File("document.txt");

// Намагаємося викликати Save() напряму - ПОМИЛКА

// myFile.Save(); // Помилка компіляції

// Виклик методу Save() через інтерфейсне посилання

ISaveable saveableFile = myFile;

saveableFile.Save();

}

}

}

1. **Приклад виконання програми**

Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст, створений ШІ, може бути неправильним.

1. **Умова завдання**

Задача 3. Інтерфейс у узагальненому класі.

Створіть узагальнений клас Box, який реалізує інтерфейс ICloneable. Метод Clone() повинен створювати копію Box.

1. **Послідовність виконання роботи**

Інтерфейс ICloneable вимагає реалізації методу Clone(), який повертає копію об’єкта.

Клас Box<T> є узагальненим (generic), тому може зберігати будь-який тип T. Він реалізує інтерфейс ICloneable та створює копію об’єкта за допомогою конструктора new Box<T>(this.Item).

Виклик методу Clone():

* Об’єкт originalBox викликає Clone(), повертаючи новий об’єкт Box<T> із тим самим вмістом.
* Щоб використовувати результат, потрібно зробити приведення типу з object до Box<string>.

1. **Код програми**

using System;

namespace GenericInterfaceExample

{

// Узагальнений клас Box<T>, який реалізує ICloneable

public class Box<T> : ICloneable

{

public T Item { get; set; }

public Box(T item)

{

Item = item;

}

// Реалізація методу Clone()

public object Clone()

{

return new Box<T>(this.Item);

}

}

class Program

{

static void Main(string[] args)

{

// Створення об’єкта Box

Box<string> originalBox = new Box<string>("Book");

// Клонування через метод Clone()

Box<string> clonedBox = (Box<string>)originalBox.Clone();

Console.WriteLine($"Original: {originalBox.Item}");

Console.WriteLine($"Clone: {clonedBox.Item}");

}

}

}

1. **Приклад виконання програми**

**Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст, створений ШІ, може бути неправильним.**