**національний університет БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ україни**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**ЛАБОРАТОРНА РОБОТА №1**

Виконав:

студент групи ІПЗ-2006б

Купчик В. Г.

Київ – 2023

**Завдання до лабораторної роботи**

**Мета роботи:** набуття вмінь і навичок роботи з конструювання ієрархії класів та використання віртуальних функцій.

**Завдання до лабораторної роботи:**

Описати базовий клас (можливо, абстрактний), у якому за допомогою віртуальних або абстрактних методів та властивостей задається інтерфейс для похідних класів.

В усіх класах потрібно перевизначити метод Equals, щоби порівнювались значення, а не посилання. Визначити в класах усі можливі конструктори.

Метод Main повинен утримувати масив або список з елементів базового класу, заповнений посиланнями на похідні класи, та демонструвати використання усіх розроблених елементів класів відповідно варіанту.

Показати на прикладі одного з методів, присутніх у кожному класі, використання поліморфізму.

Обов’язково продемонструвати різницю між раннім та пізнім зв’язуванням

Варіант - 4 (15)

Кінь, слон, тура, ферзь, шахова фігура. Поліморфний метод приймає два параметра: перший-позиція фігури на шахівниці (наприклад, g2), а другий параметр – поле, куди треба піти (наприклад, h3), і виводить повідомлення – чи можливий такий хід, чи ні;

Изображение выглядит как текст, меню, Параллельный, число

Автоматически созданное описаниеВаріант брав за даним списком

Код:

using System;

using System.Collections.Generic;

abstract class ChessPiece

{

public string Position { get; protected set; }

public ChessPiece(string position)

{

Position = position;

}

// Віртуальний метод для перевірки можливості ходу

public virtual bool CanMoveTo(string targetPosition)

{

Console.WriteLine("Базовий клас Пішка(пєшка, ті що на першій лінії): Перевірка можливості руху.");

return true;

}

// Перевизначення методу Equals для порівняння значень, а не посилань

public override bool Equals(object obj)

{

if (obj is ChessPiece other)

{

return Position == other.Position;

}

return false;

}

}

// Похідні класи: Кінь

class Knight : ChessPiece

{

public Knight(string position) : base(position) { }

public override bool CanMoveTo(string targetPosition)

{

Console.WriteLine("Кінь: Перевірка можливості руху коня на вказану позицію.");

return true;

}

}

//Слон

class Bishop : ChessPiece

{

public Bishop(string position) : base(position) { }

public override bool CanMoveTo(string targetPosition)

{

Console.WriteLine("Слон: Перевірка можливості руху слона на вказану позицію.");

return true;

}

}

//Тура

class Rook : ChessPiece

{

public Rook(string position) : base(position) { }

public override bool CanMoveTo(string targetPosition)

{

Console.WriteLine("Тура: Перевірка можливості руху тури на вказану позицію.");

return true;

}

}

//Дама

class Queen : ChessPiece

{

public Queen(string position) : base(position) { }

public override bool CanMoveTo(string targetPosition)

{

Console.WriteLine("Ферзь: Перевірка можливості руху дами на вказану позицію.");

return true;

}

}

class Program

{

static void Main()

{

//Додав укр символи

Console.OutputEncoding = System.Text.Encoding.UTF8;

// Визначення конструкторів та створення об'єктів

ChessPiece[] chessPieces = new ChessPiece[]

{

new Knight("g2"),

new Bishop("c4"),

new Rook("a1"),

new Queen("d8")

};

// Використання різних методів класів та поліморфізм

foreach (var piece in chessPieces)

{

Console.WriteLine($"Фігура на позиції {piece.Position} може здійснити хід на h3: {piece.CanMoveTo("h3")}");

}

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_early");

// Раннє зв'язування

Knight knight = new Knight("b1");

Console.WriteLine(knight.CanMoveTo("a3"));

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_late");

// Пізнє зв'язування

ChessPiece dynamicPiece = new Bishop("f5");

Console.WriteLine(dynamicPiece.CanMoveTo("d7"));

}

}

Результати виконання роботи:

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Висновок

У результаті лабораторної роботи ми розробили структуру класів для моделювання шахових фігур, навчилися використовувати поліморфізм та порівнювати об'єкти за їхніми значеннями, а також розуміємо різницю між раннім і пізнім зв'язуванням.

Контрольні питання

1. **Який модифікатор доступу дозволяє тільки членам класів-спадкоємців звертатися до поля базового класу?**

Модифікатор доступу **protected** дозволяє тільки членам класів-спадкоємців звертатися до поля базового класу.

1. **На чому базується динамічний поліморфізм?**

Динамічний поліморфізм базується на використанні віртуальних методів у базовому класі та їх перевизначенні у похідних класах. Під час виконання програми виклик методу здійснюється на основі реального типу об'єкта, що дозволяє виконувати різні реалізації методу для об'єктів різних класів.

1. **Скільки може бути базових класів у похідного класа C#?**

У мові програмування C# клас може успадковувати тільки один базовий клас (одина спадщина).

1. **Скільки об’єктів може бути створено від абстрактного класу?**

З абстрактного класу не можна створити об'єкт напряму, оскільки він призначений для використання як базовий клас для інших класів. Однак, можна створити об'єкти похідних класів, які унаслідували абстрактний клас.

1. **Чи успадковується конструктор?**

Конструктори не успадковуються від базового класу до похідного класу. Кожен клас повинен мати свій власний конструктор, і він може викликати конструктор базового класу за допомогою ключового слова **base**.

1. **Чи може бути віртуальний конструктор?**

У C# конструктори не можуть бути віртуальними. Віртуальність застосовується до методів, а не конструкторів. Проте, можна визначити віртуальні методи, які викликаються під час створення об'єкта, і перевизначити їх у похідних класах для реалізації специфічної логіки.

1. **У чому різниця між раннім та пізнім зв'язуванням?**

Раннє зв'язування відбувається під час компіляції і викликається метод на основі типу змінної (наприклад, **ChessPiece piece = new Knight();**). Пізнє зв'язування відбувається під час виконання і викликається метод на основі реального типу об'єкта (наприклад, **ChessPiece piece = new Knight();** і під час виклику **piece.CanMoveTo(...)** використовується реалізація методу у класі Knight, навіть якщо змінна оголошена як **ChessPiece**). Це дозволяє використовувати поліморфізм і обирати правильну реалізацію методу під час виконання.