ст. Гр КН-923с

Кузьмін Станіслав

**Звіт з лабораторної роботи №4**

Використання принципів та відносин об’єктно-орієнтованого програмування у мові C# .

**Мета роботи:** набути навички в написанні програм із застосуванням принципів та відносин об’єктно-орієнтованого програмування (ООП).

Варіант 6

Завдання 1

Розробити клас згідно до свого варіанта. Включити до класу методи set (…), get (…), show (…) та ін., використовуючи принцип інкапсуляції. Окрім гетерів, сетерів конструкторів та інших стандартних методів розробити не менше двох додаткових методів, які реалізують власний функціонал об’єкту відповідної предметної галузі. Написати програму, яка створює список, масив чи колекцію, об’єктів на основі відповідного класу. Під час створення використати випадковий підхід для генерації числової та текстової інформації об’єктів списку. Передбачити можливість додання, редагування та видалення об’єктів зі списку за певними критеріями. В програмі реалізувати функціонал за своїм варіантом

Train: Пункт призначення, Номер поїзда, Час відправлення, Кількість спільних місць, Купейних, Плацкартних. Створити масив об'єктів. Вивести: • список поїздів, що прямують до заданого пункту призначення; • список поїздів, що прямують до заданого пункту призначення та відправляються після заданої години; • список поїздів, що вирушають до заданого пункту призначення та мають спільні місця.

Program.cs  
using System;

using System.Collections.Generic;

class Program

{

static void Main(string[] args)

{

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

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

List<Train> trains = new List<Train>();

// Додавання випадкових поїздів до списку

Random rand = new Random();

string[] destinations = { "Kyiv", "Lviv", "Odessa", "Kharkiv", "Dnipro" };

for (int i = 0; i < 5; i++)

{

string destination = destinations[rand.Next(destinations.Length)];

int trainNumber = rand.Next(100, 999);

TimeSpan departureTime = new TimeSpan(rand.Next(0, 24), rand.Next(0, 60), 0);

int commonSeats = rand.Next(0, 100);

int compartmentSeats = rand.Next(0, 50);

int reservedSeats = rand.Next(0, 50);

trains.Add(new Train(destination, trainNumber, departureTime, commonSeats, compartmentSeats, reservedSeats));

}

// Вивід усіх поїздів

Console.WriteLine("Всі:");

foreach (var train in trains)

{

train.Show();

}

// Додавання нового поїзда

Console.WriteLine("\nДодавання нового з номером 999");

trains.Add(new Train("Zaporizhzhia", 999, new TimeSpan(14, 30, 0), 50, 30, 20));

Console.WriteLine("Всі після додавання:");

foreach (var train in trains)

{

train.Show();

}

// Редагування існуючого поїзда

Console.WriteLine("\nРедагування поїзда з номером 999...");

foreach (var train in trains)

{

if (train.TrainNumber == 999)

{

train.Destination = "Odesa"; // Використовуємо властивість

train.CommonSeats = 60; // Використовуємо властивість

}

}

Console.WriteLine("Всі після редагування:");

foreach (var train in trains)

{

train.Show();

}

// Видалення поїзда за критерієм (наприклад, поїзд до "Lviv")

Console.WriteLine("\nВидалення поїздів до Lviv...");

trains.RemoveAll(t => t.Destination == "Lviv"); // Використовуємо властивість

// Вивід списку після редагування

Console.WriteLine("\nОновлений список поїздів:");

foreach (var train in trains)

{

train.Show();

}

// Фільтрація поїздів за пунктом призначення та наявністю спільних місць

Console.WriteLine("\nВведіть пункт призначення для пошуку поїздів зі спільними місцями:");

string searchDestination = Console.ReadLine();

Console.WriteLine($"\nПоїзди до {searchDestination} зі спільними місцями:");

foreach (var train in trains)

{

if (train.Destination == searchDestination && train.HasCommonSeats()) // Використовуємо властивість

train.Show();

}

}

}

Train.cs  
using System;

public class Train

{

// Властивості класу з автоматичними гетерами та сетерами

public string Destination { get; set; } // Пункт призначення

public int TrainNumber { get; set; } // Номер поїзда

public TimeSpan DepartureTime { get; set; } // Час відправлення

public int CommonSeats { get; set; } // Кількість спільних місць

public int CompartmentSeats { get; set; } // Кількість купейних місць

public int ReservedSeats { get; set; } // Кількість плацкартних місць

// Конструктор за замовчуванням

public Train() { }

// Конструктор з параметрами

public Train(string destination, int trainNumber, TimeSpan departureTime, int commonSeats, int compartmentSeats, int reservedSeats)

{

// Ініціалізація властивостей

Destination = destination;

TrainNumber = trainNumber;

DepartureTime = departureTime;

CommonSeats = commonSeats;

CompartmentSeats = compartmentSeats;

ReservedSeats = reservedSeats;

}

// Метод show для відображення інформації про поїзд

public void Show()

{

Console.WriteLine($"\nTrain Number: {TrainNumber}, Destination: {Destination}, Departure Time: {DepartureTime}, Common Seats: {CommonSeats}, Compartment Seats: {CompartmentSeats}, Reserved Seats: {ReservedSeats}");

}

// Метод для перевірки наявності спільних місць

public bool HasCommonSeats()

{

return CommonSeats > 0;

}

// Додатковий метод: Перевірка, чи поїзд відправляється після заданої години

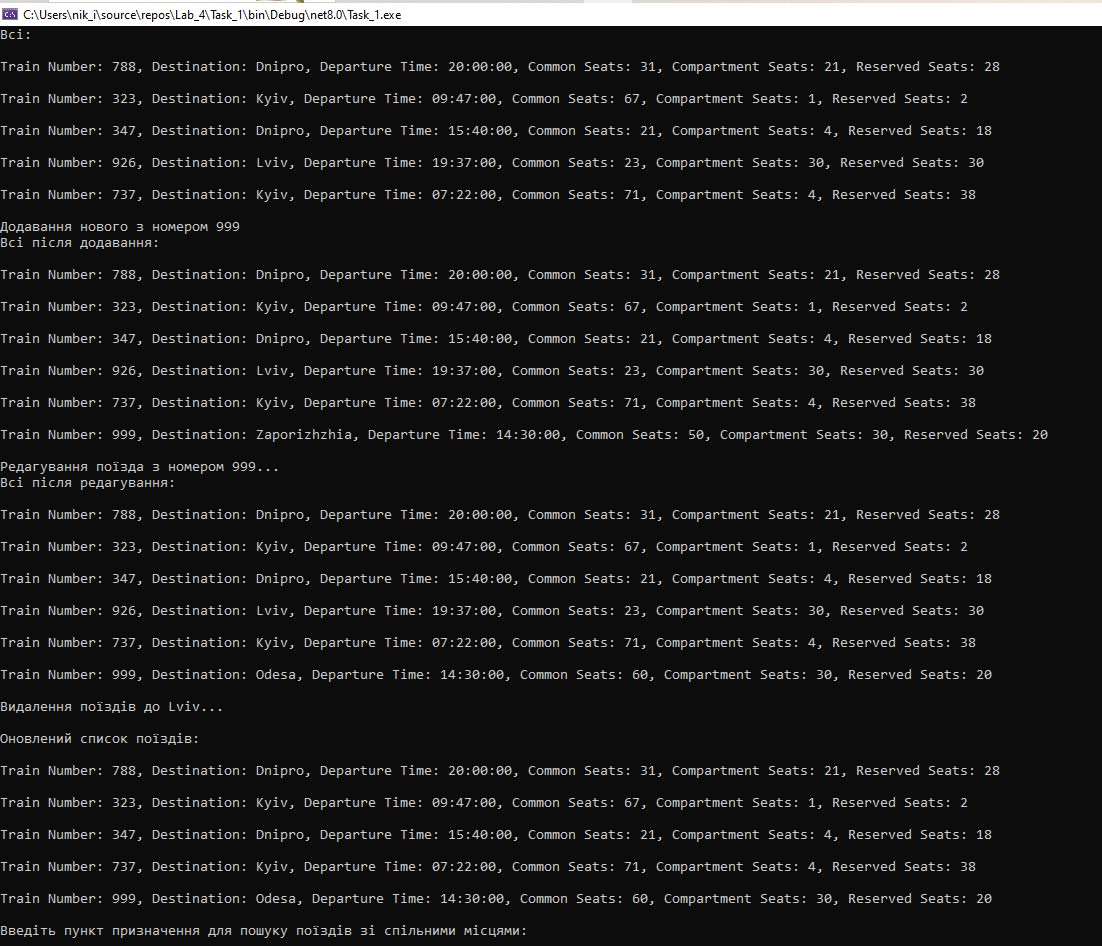
public bool DepartsAfter(TimeSpan time)

{

return DepartureTime > time;

}

}



Завдання 2

Program.cs

using System;

using System.Collections.Generic;

public class Program

{

public static void Main(string[] args)

{

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

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

// Створюємо список поліморфних об'єктів типу Transport

List<Transport> transports = new List<Transport>();

// Додаємо об'єкти різних класів

transports.Add(new Train("Kyiv", 123, new TimeSpan(12, 30, 0), 100, 50, 20));

transports.Add(new Plane("Lviv", 456, new TimeSpan(15, 45, 0), "Ukrainian Airlines", 12, 120));

transports.Add(new Bus("Odesa", 789, new TimeSpan(10, 0, 0), 50, true));

transports.Add(new LuxuryBus("Kharkiv-Lux", 321, new TimeSpan(14, 15, 0), 30, true, true, true));

transports.Add(new ElectricBus("Chernivtsi-Electric", 654, new TimeSpan(9, 0, 0), 40, false, 100, 300));

// Викликаємо метод Show для всіх об'єктів у списку

foreach (Transport transport in transports)

{

transport.Show();

}

}

}

Trainsport.cs

// Абстрактний базовий клас

public abstract class Transport

{

// Абстрактні властивості

public abstract string Destination { get; set; }

public abstract int TrainNumber { get; set; }

public abstract TimeSpan DepartureTime { get; set; }

// Абстрактний метод

public abstract void Show();

// Конкретний метод

public virtual bool DepartsAfter(TimeSpan time)

{

return DepartureTime > time;

}

}

Train.cs  
using System;

// Клас Train, що наслідується від Transport

public class Train : Transport

{

// Реалізація властивостей

public override string Destination { get; set; }

public override int TrainNumber { get; set; }

public override TimeSpan DepartureTime { get; set; }

public int CommonSeats { get; set; }

public int CompartmentSeats { get; set; }

public int ReservedSeats { get; set; }

// Конструктор

public Train(string destination, int trainNumber, TimeSpan departureTime, int commonSeats, int compartmentSeats, int reservedSeats)

{

Destination = destination;

TrainNumber = trainNumber;

DepartureTime = departureTime;

CommonSeats = commonSeats;

CompartmentSeats = compartmentSeats;

ReservedSeats = reservedSeats;

}

// Реалізація абстрактного методу

public override void Show()

{

Console.ForegroundColor = ConsoleColor.DarkMagenta;

Console.Write("Train - ");

Console.ResetColor();

Console.WriteLine($"Номер поїзда: {TrainNumber}, Пункт призначення: {Destination}, Час відправлення: {DepartureTime}, Спільні місця: {CommonSeats}, Купейні місця: {CompartmentSeats}, Плацкартні місця: {ReservedSeats}");

}

// Метод для перевірки наявності спільних місць

public bool HasCommonSeats()

{

return CommonSeats > 0;

}

}

Bus.cs

using System;

public class Bus : Transport

{

// Реалізація абстрактних властивостей

public override string Destination { get; set; }

public override int TrainNumber { get; set; }

public override TimeSpan DepartureTime { get; set; }

public int Seats { get; set; } // Кількість місць в автобусі

public bool HasWiFi { get; set; } // Наявність WiFi

public Bus(string destination, int busNumber, TimeSpan departureTime, int seats, bool hasWiFi)

{

Destination = destination;

TrainNumber = busNumber;

DepartureTime = departureTime;

Seats = seats;

HasWiFi = hasWiFi;

}

public override void Show()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.Write("Bus - ");

Console.ResetColor();

Console.WriteLine($"Bus Number: {TrainNumber}, Destination: {Destination}, Departure Time: {DepartureTime}, Seats: {Seats}, WiFi: {(HasWiFi ? "Yes" : "No")}");

}

}

ElectricBus.cs  
using System;

public class ElectricBus : Bus

{

public int BatteryCapacity { get; set; } // Ємність батареї в кВт·год

public int Range { get; set; } // Дальність поїздки на одному заряді в км

public ElectricBus(string destination, int busNumber, TimeSpan departureTime, int seats, bool hasWiFi, int batteryCapacity, int range)

: base(destination, busNumber, departureTime, seats, hasWiFi)

{

BatteryCapacity = batteryCapacity;

Range = range;

}

public void ChargeBattery() // Новий метод

{

if (BatteryCapacity <= 0) // Перевірка на нульову ємність

{

Console.WriteLine("Не можна заряджати: ємність батареї дорівнює нулю.");

}

else

{

Console.WriteLine("Батарея заряджається...");

}

}

public void DisplayRange() // Новий метод

{

Console.WriteLine($"Дальність поїздки на одному заряді: {Range} км.");

}

public override void Show()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.Write("Electric ");

Console.ResetColor();

base.Show();

Console.WriteLine($"Battery Capacity: {BatteryCapacity} kWh, Range: {Range} km");

}

}

LuxuryBus.cs

using System;

public class LuxuryBus : Bus

{

public bool HasToilet { get; set; }

public bool HasRecliningSeats { get; set; }

public LuxuryBus(string destination, int busNumber, TimeSpan departureTime, int seats, bool hasWiFi, bool hasToilet, bool hasRecliningSeats)

: base(destination, busNumber, departureTime, seats, hasWiFi)

{

HasToilet = hasToilet;

HasRecliningSeats = hasRecliningSeats;

}

public void ServeDrinks() // Новий метод

{

Console.WriteLine("Напої подаються пасажирам...");

}

public void ProvideEntertainment() // Новий метод

{

Console.WriteLine("Пасажирам надаються розважальні програми...");

}

public override void Show()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.Write("Luxury ");

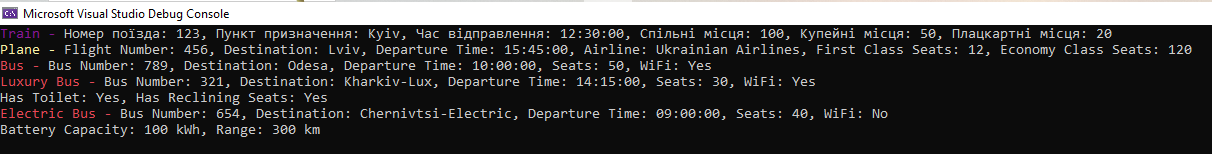
Console.ResetColor();

base.Show();

Console.WriteLine($"Has Toilet: {(HasToilet ? "Yes" : "No")}, Has Reclining Seats: {(HasRecliningSeats ? "Yes" : "No")}");

}

}



Завдання 3

Розробити клас та перевантажити оператори згідно до свого варіанта. В головній функції програми протестувати роботу створених класів на прикладі використання окремих об’єктів та масивів чи колекцій цих об’єктів. Під час створення об’єктів застосувати випадковий підхід чи зчитування інформації з файлів.

1. Визначити клас "Стек" - Stack. Елементи стека зберігаються у масиві. Якщо масив має фіксовану розмірність, передбачити контроль виходу межі масиву. Якщо пам'ять виділяється динамічно і не вистачає, то збільшити розмір виділеної пам'яті. Включення елементів у стек та їх вилучення реалізувати шляхом перевантажених відповідних операторів. Створити масив об'єктів класу Stack. Передавати об'єкти у функцію, яка видаляє зі стека перший (згори), третій, п'ятий тощо елементи

Program.cs

using System;

public class Program

{

public static void Main(string[] args)

{

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

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

// Створюємо новий об'єкт стека

Stack stack = new Stack();

// Додаємо елементи до стека за допомогою перевантаженого оператора +

stack += 1; // Додаємо 1

stack += 2; // Додаємо 2

stack += 3; // Додаємо 3

stack += 4; // Додаємо 4

stack += 5; // Додаємо 5

// Виводимо вміст стека перед видаленням елементів

Console.WriteLine("Стек:");

stack.PrintStack(); // Викликаємо метод для виводу вмісту стека

int[] indicesToRemove = { 1, 3 }; // Видалення 1-й (значення 2) і 3-й (значення 4) елеменів

// Виклик метода для видалення елементів з заданими індексами

stack.RemoveElements(indicesToRemove);

// Вивід вміст стека після видалення елементів

Console.WriteLine("Після видалення:");

stack.PrintStack(); // метод для виводу вмісту стека

}

}

Stack.cs

using System;

public class Stack

{

// Масив для зберігання елементів стека

private int[] \_elements;

// Кількість елементів у стеку

private int \_size;

// Максимальна місткість стека

private int \_capacity;

// Конструктор за замовчуванням

public Stack(int capacity = 10)

{

\_capacity = capacity; // Ініціалізація максимального розміру стека

\_elements = new int[\_capacity]; // Виділення пам'яті під масив

\_size = 0; // Ініціалізація кількості елементів

}

// Перевантажений оператор для додавання елемента в стек

public static Stack operator +(Stack stack, int value)

{

// Перевіряємо, чи стек переповнений

if (stack.\_size >= stack.\_capacity)

{

// Якщо переповнений, збільшуємо його розмір

stack.Resize();

}

// Додаємо новий елемент у верхню частину стека і збільшуємо розмір

stack.\_elements[stack.\_size++] = value;

return stack; // Повертаємо оновлений стек

}

// Перевантажений оператор для видалення елемента зі стека

public static Stack operator -(Stack stack)

{

// Перевіряємо, чи стек не пустий

if (stack.\_size > 0)

{

// Зменшуємо розмір стека, видаляючи верхній елемент

stack.\_size--;

}

return stack; // Повертаємо оновлений стек

}

// Метод для зміни розміру масиву

private void Resize()

{

\_capacity \*= 2; // Подвоюємо місткість стека

int[] newElements = new int[\_capacity]; // Створюємо новий масив з новою місткістю

// Копіюємо старі елементи в новий масив

Array.Copy(\_elements, newElements, \_size);

\_elements = newElements; // Заміщуємо старий масив новим

}

// Метод для отримання значення верхнього елемента стека

public int Peek()

{

// Перевіряємо, чи стек не пустий

if (\_size > 0)

return \_elements[\_size - 1]; // Повертаємо верхній елемент

// Якщо стек пустий, викидаємо виключення

throw new InvalidOperationException("Stack is empty.");

}

// Метод для видалення елементів за заданими індексами

public void RemoveElements(int[] indices)

{

// Проходимо по масиву індексів, які потрібно видалити

for (int i = 0; i < indices.Length; i++)

{

// Перевіряємо, чи індекс дійсний

if (indices[i] < \_size && indices[i] >= 0)

{

// Зсув елементів для видалення

for (int j = indices[i]; j < \_size - 1; j++)

{

// Копіюємо наступний елемент на позицію поточного

\_elements[j] = \_elements[j + 1];

}

// Зменшуємо кількість елементів у стека

\_size--;

}

}

}

// Метод для виводу елементів стека в консоль

public void PrintStack()

{

Console.WriteLine("Stack elements:");

// Проходимо по всім елементам стека

for (int i = 0; i < \_size; i++)

{

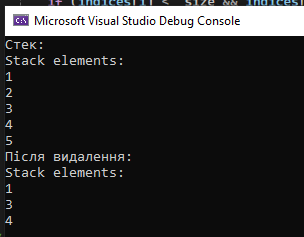
// Виводимо кожен елемент

Console.WriteLine(\_elements[i]);

}

}

}



Завдання 4

Агрегація

Головний клас - Людина

Підпорядкований клас - Майно

**Program.cs**

using System;

class Program

{

static void Main(string[] args)

{

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

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

Person person = new Person("Іван");

Property house = new Property("Будинок", 150000);

Property car = new Property("Автомобіль", 25000);

person.AddProperty(house);

person.AddProperty(car);

person.ShowProperties();

person.RemoveProperty(car);

person.ShowProperties();

}

}

**Person.cs**

using System.Collections.Generic;

// Клас для представлення особи

public class Person

{

// Властивість для зберігання імені особи

public string Name { get; set; }

// Список, що містить майно, яке належить цій особі

public List<Property> properties;

// Конструктор класу, який приймає ім'я особи

public Person(string name)

{

Name = name; // Присвоюємо ім'я особи

properties = new List<Property>(); // Ініціалізуємо список майна

}

// Метод для додавання майна до списку

public void AddProperty(Property property)

{

properties.Add(property); // Додаємо майно до списку

Console.WriteLine($"Додано майно: {property}"); // Виводимо підтвердження

}

// Метод для видалення майна зі списку

public void RemoveProperty(Property property)

{

// Спробуємо видалити майно з списку

if (properties.Remove(property))

{

Console.WriteLine($"Видалено майно: {property}"); // Виводимо підтвердження

}

else

{

Console.WriteLine($"Майно не знайдено: {property}"); // Виводимо повідомлення, якщо майно не знайдено

}

}

// Метод для відображення майна особи

public void ShowProperties()

{

Console.WriteLine($"{Name} має наступне майно:"); // Виводимо заголовок

foreach (var property in properties) // Проходимося по списку майна

{

Console.WriteLine(property); // Виводимо інформацію про кожне майно

}

}

}

**Property.cs**

// Клас для представлення майна

public class Property

{

// Властивість для зберігання назви майна

public string Name { get; set; }

// Властивість для зберігання вартості майна

public decimal Value { get; set; }

// Конструктор класу, який приймає назву і вартість майна

public Property(string name, decimal value)

{

Name = name; // Присвоюємо назву майна

Value = value; // Присвоюємо вартість майна

}

// Перевизначення методу ToString() для отримання текстового представлення майна

public override string ToString()

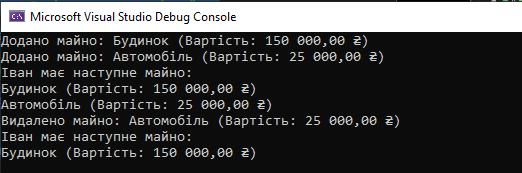
{

// Повертаємо рядок, що містить назву та вартість майна, відформатовану як валюта

return $"{Name} (Вартість: {Value:C})";

}

}



Композиція

Головний клас - Людина

Підпорядкований клас - Талант

**Program.cs**

using System;

class Program

{

static void Main(string[] args)

{

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

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

// Створення об'єкта людини

Person person = new Person("Андрій");

// Створення навичок

Skill skill1 = new Skill("Програмування", 8);

Skill skill2 = new Skill("Гра на гітарі", 6);

// Додавання навичок до людини

person.AddSkill(skill1);

person.AddSkill(skill2);

// Показ навичок

person.ShowSkills();

// Видалення навички

person.RemoveSkill(skill1);

// Показ навичок після видалення

person.ShowSkills();

}

}

**Person.cs**

using System;

using System.Collections.Generic;

public class Person

{

public string Name { get; set; }

private List<Skill> skills; // Композиційний зв'язок

public Person(string name)

{

Name = name;

skills = new List<Skill>(); // Ініціалізація списку навичок

}

public void AddSkill(Skill skill)

{

skills.Add(skill);

Console.WriteLine($"Додано навичку: {skill}");

}

public void RemoveSkill(Skill skill)

{

if (skills.Remove(skill))

{

Console.WriteLine($"Видалено навичку: {skill}");

}

else

{

Console.WriteLine($"Навичка не знайдена: {skill}");

}

}

public void ShowSkills()

{

Console.WriteLine($"{Name} має наступні навички:");

foreach (var skill in skills)

{

Console.WriteLine(skill);

}

}

}

**Skill.cs**

using System;

public class Skill

{

public string Name { get; set; }

public int Proficiency { get; set; } // Від 1 до 10

public Skill(string name, int proficiency)

{

Name = name;

Proficiency = proficiency;

}

public override string ToString()

{

return $"{Name} (Рівень: {Proficiency})";

}

}

