Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

**Звіт**

**Лабораторна робота 3**

з дисципліни "Кросплатформне програмування"

Виконав: студент групи КН-1224С

Мульков М. В.

Перевірив:

Івашко А.В.

Харків 2025

Зміст

[Завдання 3](#_Toc193543854)

[Реалізація коду 4](#_Toc193543855)

[Алгоритм коду 4](#_Toc193543856)

[Тестування коду 6](#_Toc193543857)

[Висновок 8](#_Toc193543858)

[Додаток А 10](#_Toc193543859)

**Робота з успадкуванням класів у .NET Framework**

**Мета роботи**: Дослідити роботу успадкування класів, навчиться створювати ієрархію успадкування, виконати перевантаження функцій у класах.

# Завдання

В рамках даної роботи необхідно:

Створити базовий клас Product із властивостями:

string Name – ім'я товару;

decimal Price – вартість товару.

Розробити декілька похідних класів за мінімум трьома рівнями успадкування. Для кожного рівня має бути щонайменше два класи, які відрізняються набором властивостей відповідно до специфіки товару. Набір додаткових властивостей узгоджується з викладачем.

Для кожного класу реалізувати конструктор за замовчуванням, а також кілька конструкторів з різними параметрами, що дозволить створювати об’єкти із різними наборами даних.

Перевантажити метод ToString() у всіх класах, щоб забезпечити можливість перетворення інформації про об’єкт у зручний для виведення рядок.

Розробити інтерактивну консольну програму, яка дозволить тестувати створені класи. Програма повинна працювати у безкінечному циклі, надаючи користувачу можливість вибрати один із двох режимів роботи:

Режим автоматичного створення об’єктів із випадковими параметрами.

Режим ручного введення параметрів для створення об’єктів.

Забезпечити кольорове відображення інформації у консолі, що полегшує розрізнення типів об’єктів та різних розділів меню.

# Реалізація коду

## Алгоритм коду

Повний код програми відображено у додатку А.

1. Виведення головного меню

Після запуску програми користувачу відображається головне меню, оформлене кольоровими шрифтами.

Меню містить три варіанти: тест з випадковим створенням об’єктів, тест з ручним введенням параметрів, а також опцію для виходу з програми. Користувач вводить свій вибір, і програма перевіряє коректність введення. У разі некоректного вибору повідомлення про помилку виводиться червоним кольором.

Програма працює в безкінечному циклі, що дозволяє користувачу повторювати тестування або переходити між режимами. Вибір опції «Вихід» припиняє виконання програми.

2. Режим автоматичного створення об’єктів

При виборі цього режиму програма переходить до етапу генерації параметрів.

Використовуються заздалегідь визначені масиви, що містять типові значення для назв, локацій, типів товарів, інвестицій та ґрунтів. За допомогою генератора випадкових чисел програма обирає значення із цих масивів.

Для кожного класу (Product, RealEstate, RealEstateInvestment, Apartment, House, Hotel, LandPlot) створюється об’єкт із випадковим набором параметрів. Наприклад, для класу Product генерується назва з масиву та випадкова вартість у визначеному діапазоні.

Після створення кожного об’єкту його інформація виводиться на екран із застосуванням відповідного кольору, що робить вивід наочним. Межі між блоками інформації відокремлюються додатковими порожніми рядками.

3. Режим ручного введення

При виборі цього режиму користувачу пропонується окреме меню з переліком типів об’єктів, які можна створити. Після вибору типу об’єкту користувача покроково запрошують ввести необхідні дані (наприклад, назву, ціну, локацію, розмір, тип та інші параметри залежно від обраного класу).

Після введення даних створюється відповідний об’єкт, і його інформація виводиться на екран у кольоровому форматі. Для кожного типу об’єкта використовується окремий колір, що дозволяє швидко орієнтуватися у результатах тестування.

Програма після завершення ручного введення повертає користувача до головного меню.

## Тестування коду

Тестування коду з автоматичним створенням відображено на рис. 1.



Рис. 1. Тестування коду з автоматичним створенням

Тестування коду з ручним введенням відображено на рис. 2.

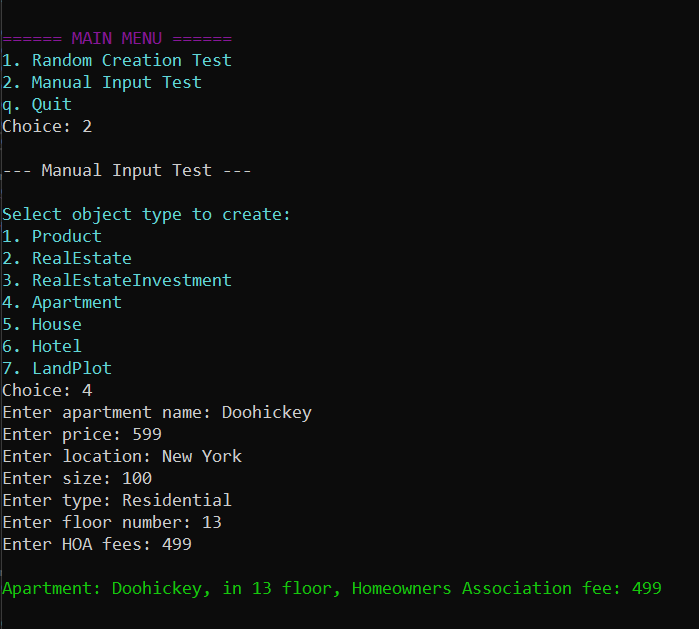


Рис. 2. Тестування коду з ручним введенням

# Висновок

У ході виконання даної роботи було детально опрацьовано принципи побудови ієрархії класів для зберігання та обробки інформації. Було засвоєно основні підходи об’єктно-орієнтованого програмування, зокрема наслідування, інкапсуляцію та поліморфізм, що дозволило створити базовий клас Product із загальними властивостями та розширити його за допомогою декількох похідних класів із додатковими параметрами. Особлива увага приділена перевантаженню методу ToString(), що забезпечує уніфіковане та зручне відображення інформації про об’єкти різних типів.

У розробленій програмі реалізовано два режими тестування – автоматичне створення об’єктів із випадковими параметрами та режим ручного введення даних користувачем. Завдяки використанню попередньо заданих масивів значень для генерації параметрів було досягнуто високої гнучкості та варіативності створених тестових прикладів. Інтерактивне головне меню з кольоровим відображенням значно полегшує роботу користувача, дозволяючи швидко перемикатися між режимами та отримувати наочний результат.

Крім того, програму характеризує динамічна структура тестування, що забезпечує можливість повторного запуску операцій без необхідності перезавантаження додатка. Завдяки чіткому розподілу логіки на етапи – виведення меню, генерація випадкових параметрів, створення об’єктів, їх відображення та обробка введених даних – була реалізована система, що є як функціональною, так і масштабованою для подальшого розширення.

Таким чином, виконання роботи дозволило не лише закріпити теоретичні знання з об’єктно-орієнтованого програмування, а й на практиці продемонструвати ефективні методи розробки програмних систем для роботи з різноманітними даними.

# Додаток А

namespace lb\_3;

class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

public Product()

{

Name = string.Empty;

Price = 0;

}

public Product(string name, decimal price)

{

Name = name;

Price = price;

}

public virtual string ToString()

{

return $"{Name}, Price: {Price}";

}

}

class RealEstate : Product

{

public string Location { get; set; }

public double Size { get; set; }

public string Type { get; set; }

public RealEstate()

{

Location = string.Empty;

Size = 0;

Type = string.Empty;

}

public RealEstate(string location, double size)

{

Location = location;

Size = size;

Type = string.Empty;

}

public RealEstate(string location, double size, string type)

{

Location = location;

Size = size;

Type = type;

}

public RealEstate(string name, decimal price, string location, double size, string type)

: base(name, price)

{

Location = location;

Size = size;

Type = type;

}

public override string ToString()

{

return $"{Name}, Price: {Price}, Location: {Location}";

}

}

class RealEstateInvestment : Product

{

public string Location { get; set; }

public decimal MarketValue { get; set; }

public string InvestmentType { get; set; }

public RealEstateInvestment()

{

Location = string.Empty;

MarketValue = 0;

InvestmentType = string.Empty;

}

public RealEstateInvestment(string location, decimal marketValue)

{

Location = location;

MarketValue = marketValue;

InvestmentType = string.Empty;

}

public RealEstateInvestment(string location, decimal marketValue, string investmentType)

{

Location = location;

MarketValue = marketValue;

InvestmentType = investmentType;

}

public RealEstateInvestment(string name, decimal price, string location, decimal marketValue, string investmentType)

: base(name, price)

{

Location = location;

MarketValue = marketValue;

InvestmentType = investmentType;

}

public override string ToString()

{

return $"{Name}, Location: {Location}, Value: {MarketValue}, Invenstment type: {InvestmentType}";

}

}

class Apartment : RealEstate

{

public int FloorNumber { get; set; }

public decimal HOAFees { get; set; }

public Apartment()

{

FloorNumber = 0;

HOAFees = 0;

}

public Apartment(int floorNumber, decimal fees)

{

FloorNumber = floorNumber;

HOAFees = fees;

}

public Apartment(string name, decimal price, string location, double size, string type, int floorNumber, decimal fees)

: base(name, price, location, size, type)

{

FloorNumber = floorNumber;

HOAFees = fees;

}

public override string ToString()

{

return $"{Name}, in {FloorNumber} floor, Homeowners Association fee: {HOAFees}";

}

}

class House : RealEstate

{

public double GardenSize { get; set; }

public bool Pool { get; set; }

public House()

{

GardenSize = 0;

Pool = false;

}

public House(double gardenSize, bool pool)

{

GardenSize = gardenSize;

Pool = pool;

}

public House(string name, decimal price, string location, double size, string type, double gardenSize, bool pool)

: base(name, price, location, size, type)

{

GardenSize = gardenSize;

Pool = pool;

}

public override string ToString()

{

return $"{Name}, garden size {GardenSize}, {(Pool ? "there is" : "no")} pool";

}

}

class Hotel : RealEstateInvestment

{

public int Rooms { get; set; }

public int StartRating { get; set; }

public Hotel()

{

Rooms = 0;

StartRating = 0;

}

public Hotel(int rooms, int startRating)

{

Rooms = rooms;

StartRating = startRating;

}

public Hotel(string name, decimal price, string location, decimal marketValue, string investmentType, int rooms, int startRating)

: base(name, price, location, marketValue, investmentType)

{

Rooms = rooms;

StartRating = startRating;

}

public override string ToString()

{

return $"{Name}, there are {Rooms} rooms, Hotel rating {StartRating}";

}

}

class LandPlot : RealEstateInvestment

{

public string SoilType { get; set; }

public bool InfrastructureAccess { get; set; }

public LandPlot()

{

SoilType = string.Empty;

InfrastructureAccess = true;

}

public LandPlot(string soilType, bool infrastructureAccess)

{

SoilType = soilType;

InfrastructureAccess = infrastructureAccess;

}

public LandPlot(string name, decimal price, string location, decimal marketValue, string investmentType, string soilType, bool infrastructureAccess)

: base(name, price, location, marketValue, investmentType)

{

SoilType = soilType;

InfrastructureAccess = infrastructureAccess;

}

public override string ToString()

{

return $"{Name}, Soil type {SoilType}, {(InfrastructureAccess ? "have": "no")} access to infrastructure";

}

}

class Program

{

static Random rnd = new Random();

static string[] prodNames = { "Widget", "Gadget", "Thingamajig", "Doohickey", "Contraption" };

static string[] reNames = { "Luxury Villa", "Modern Condo", "Country House", "Beachfront Estate", "Urban Loft" };

static string[] locations = { "Beverly Hills", "New York", "Miami", "Los Angeles", "Chicago" };

static string[] reTypes = { "Residential", "Commercial", "Industrial" };

static string[] invTypes = { "Commercial", "Residential", "Mixed-Use" };

static string[] soilTypes = { "Loamy", "Sandy", "Clay", "Silty" };

static void Main()

{

while (true)

{

Console.ResetColor();

Console.ForegroundColor = ConsoleColor.DarkMagenta;

Console.WriteLine("\n====== MAIN MENU ======");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("1. Random Creation Test");

Console.WriteLine("2. Manual Input Test");

Console.WriteLine("q. Quit");

Console.ResetColor();

Console.Write("Choice: ");

string choice = Console.ReadLine();

if (choice.ToLower() == "q") break;

if (choice == "1") RandomTest();

else if (choice == "2") ManualTest();

else { Console.ForegroundColor = ConsoleColor.Red; Console.WriteLine("\nInvalid choice. Try again.\n"); }

}

}

static void RandomTest()

{

Console.ResetColor();

Console.WriteLine("\n--- Random Creation Test ---\n");

Console.ForegroundColor = ConsoleColor.Yellow;

var prod = new Product(

prodNames[rnd.Next(prodNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 1000));

Console.WriteLine("Product: " + prod.ToString() + "\n");

Console.ForegroundColor = ConsoleColor.Cyan;

var re = new RealEstate(

reNames[rnd.Next(reNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 500000),

locations[rnd.Next(locations.Length)],

rnd.Next(500, 10000),

reTypes[rnd.Next(reTypes.Length)]);

Console.WriteLine("RealEstate: " + re.ToString() + "\n");

Console.ForegroundColor = ConsoleColor.Magenta;

var rei = new RealEstateInvestment(

"Investment " + prodNames[rnd.Next(prodNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 500000),

locations[rnd.Next(locations.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 1000000),

invTypes[rnd.Next(invTypes.Length)]);

Console.WriteLine("RealEstateInvestment: " + rei.ToString() + "\n");

Console.ForegroundColor = ConsoleColor.Green;

var apt = new Apartment(

"Apartment " + reNames[rnd.Next(reNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 300000),

locations[rnd.Next(locations.Length)],

rnd.Next(300, 1500),

"Condo",

rnd.Next(1, 50),

Convert.ToDecimal(rnd.NextDouble() \* 500));

Console.WriteLine("Apartment: " + apt.ToString() + "\n");

Console.ForegroundColor = ConsoleColor.Blue;

var house = new House(

"House " + reNames[rnd.Next(reNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 450000),

locations[rnd.Next(locations.Length)],

rnd.Next(500, 3000),

"Detached",

rnd.Next(50, 1000),

rnd.Next(0, 2) == 0);

Console.WriteLine("House: " + house.ToString() + "\n");

Console.ForegroundColor = ConsoleColor.DarkYellow;

var hotel = new Hotel(

"Hotel " + reNames[rnd.Next(reNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 1000000),

locations[rnd.Next(locations.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 2000000),

invTypes[rnd.Next(invTypes.Length)],

rnd.Next(10, 200),

rnd.Next(1, 6));

Console.WriteLine("Hotel: " + hotel.ToString() + "\n");

Console.ForegroundColor = ConsoleColor.DarkCyan;

var land = new LandPlot(

"Land " + prodNames[rnd.Next(prodNames.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 1000000),

locations[rnd.Next(locations.Length)],

Convert.ToDecimal(rnd.NextDouble() \* 1500000),

invTypes[rnd.Next(invTypes.Length)],

soilTypes[rnd.Next(soilTypes.Length)],

rnd.Next(0, 2) == 1);

Console.WriteLine("LandPlot: " + land.ToString() + "\n");

Console.ResetColor();

}

static void ManualTest()

{

Console.ResetColor();

Console.WriteLine("\n--- Manual Input Test ---\n");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("Select object type to create:");

Console.WriteLine("1. Product");

Console.WriteLine("2. RealEstate");

Console.WriteLine("3. RealEstateInvestment");

Console.WriteLine("4. Apartment");

Console.WriteLine("5. House");

Console.WriteLine("6. Hotel");

Console.WriteLine("7. LandPlot");

Console.ResetColor();

Console.Write("Choice: ");

string choice = Console.ReadLine();

switch (choice)

{

case "1":

{

Console.Write("Enter product name: ");

string name = Console.ReadLine();

Console.Write("Enter product price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

var p = new Product(name, price);

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("\nProduct: " + p.ToString() + "\n");

break;

}

case "2":

{

Console.Write("Enter real estate name: ");

string name = Console.ReadLine();

Console.Write("Enter price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter location: ");

string loc = Console.ReadLine();

Console.Write("Enter size: ");

double size = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter type: ");

string type = Console.ReadLine();

var re = new RealEstate(name, price, loc, size, type);

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("\nRealEstate: " + re.ToString() + "\n");

break;

}

case "3":

{

Console.Write("Enter investment name: ");

string name = Console.ReadLine();

Console.Write("Enter price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter location: ");

string loc = Console.ReadLine();

Console.Write("Enter market value: ");

decimal marketValue = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter investment type: ");

string invType = Console.ReadLine();

var rei = new RealEstateInvestment(name, price, loc, marketValue, invType);

Console.ForegroundColor = ConsoleColor.Magenta;

Console.WriteLine("\nRealEstateInvestment: " + rei.ToString() + "\n");

break;

}

case "4":

{

Console.Write("Enter apartment name: ");

string name = Console.ReadLine();

Console.Write("Enter price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter location: ");

string loc = Console.ReadLine();

Console.Write("Enter size: ");

double size = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter type: ");

string type = Console.ReadLine();

Console.Write("Enter floor number: ");

int floor = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter HOA fees: ");

decimal fees = Convert.ToDecimal(Console.ReadLine());

var apt = new Apartment(name, price, loc, size, type, floor, fees);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nApartment: " + apt.ToString() + "\n");

break;

}

case "5":

{

Console.Write("Enter house name: ");

string name = Console.ReadLine();

Console.Write("Enter price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter location: ");

string loc = Console.ReadLine();

Console.Write("Enter size: ");

double size = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter type: ");

string type = Console.ReadLine();

Console.Write("Enter garden size: ");

double garden = Convert.ToDouble(Console.ReadLine());

Console.Write("Has pool? (true/false): ");

bool pool = Convert.ToBoolean(Console.ReadLine());

var house = new House(name, price, loc, size, type, garden, pool);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\nHouse: " + house.ToString() + "\n");

break;

}

case "6":

{

Console.Write("Enter hotel name: ");

string name = Console.ReadLine();

Console.Write("Enter price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter location: ");

string loc = Console.ReadLine();

Console.Write("Enter market value: ");

decimal marketValue = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter investment type: ");

string invType = Console.ReadLine();

Console.Write("Enter number of rooms: ");

int rooms = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter star rating: ");

int rating = Convert.ToInt32(Console.ReadLine());

var hotel = new Hotel(name, price, loc, marketValue, invType, rooms, rating);

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("\nHotel: " + hotel.ToString() + "\n");

break;

}

case "7":

{

Console.Write("Enter land plot name: ");

string name = Console.ReadLine();

Console.Write("Enter price: ");

decimal price = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter location: ");

string loc = Console.ReadLine();

Console.Write("Enter market value: ");

decimal marketValue = Convert.ToDecimal(Console.ReadLine());

Console.Write("Enter investment type: ");

string invType = Console.ReadLine();

Console.Write("Enter soil type: ");

string soil = Console.ReadLine();

Console.Write("Infrastructure access (true/false): ");

bool access = Convert.ToBoolean(Console.ReadLine());

var land = new LandPlot(name, price, loc, marketValue, invType, soil, access);

Console.ForegroundColor = ConsoleColor.DarkCyan;

Console.WriteLine("\nLandPlot: " + land.ToString() + "\n");

break;

}

default:

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("\nInvalid selection. Try again.\n");

break;

}

}

Console.ResetColor();

}

}