Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

**Звіт**

**Лабораторна робота 4**

з дисципліни "Кросплатформне програмування"

Виконав: студент групи КН-1224С

Мульков М. В.

Перевірив:

Івашко А.В.

Харків 2025

Зміст

[Завдання 3](#_Toc193835966)

[Реалізація коду 4](#_Toc193835967)

[Алгоритм коду 4](#_Toc193835968)

[Тестування коду 6](#_Toc193835969)

[Висновок 8](#_Toc193835970)

[Додаток А 9](#_Toc193835971)

**Робота з** **класом-контейнером у .NET Framework**

**Мета роботи**: Розробити клас-контейнер для зберігання об'єктів розробленої ієрархії класів, навчитися реалізовувати базові операції над контейнером (додавання, видалення, упорядкування) без використання стандартних методів класу Array.

# Завдання

В рамках даної роботи необхідно:

Створити клас Container, який є оболонкою над масивом для зберігання об'єктів ієрархії товарів (з попередніх лабораторних робіт).

Реалізувати метод Add() для додавання об'єкта до контейнера. Передбачити автоматичне розширення масиву при необхідності.

Реалізувати метод RemoveById() для видалення об'єкта з контейнера за вказаним індексом.

Реалізувати метод Sort() для впорядкування об'єктів у контейнері. Реалізувати алгоритм сортування самостійно.

Реалізувати метод ToString() для перетворення вмісту контейнера у рядок для виведення на консоль.

Розробити інтерактивну консольну програму, яка дозволяє тестувати функціональність класу Container: додавання, видалення, сортування та виведення вмісту контейнера.

# Реалізація коду

## Алгоритм коду

Повний код програми відображено у додатку А.

1. Створення класу Container

Клас Container реалізовано як оболонка над масивом items типу object[]. Приватні поля count та size використовуються для відстеження поточної кількості елементів та розміру масиву відповідно. Конструктор за замовчуванням ініціалізує масив початковим розміром в 1 елемент.

2. Реалізація методу Add()

Метод Add додає новий об'єкт \_newObject до контейнера. Якщо масив заповнений (count == size), відбувається його автоматичне розширення: створюється новий масив вдвічі більшого розміру, куди копіюються існуючі елементи, і масив items замінюється новим. Після цього новий об'єкт додається в кінець масиву, і значення count збільшується.

3. Реалізація методу RemoveById()

Метод RemoveById видаляє об'єкт за вказаним індексом \_index. Спочатку виконується перевірка на коректність індексу. Якщо індекс виходить за межі масиву, генерується виняток IndexOutOfRangeException. Для видалення елемента створюється новий масив newArray того ж розміру. Елементи з вихідного масиву копіюються до нового масиву, пропускаючи елемент за вказаним індексом. Масив items замінюється новим масивом, а count зменшується. Метод повертає видалений об'єкт.

4. Реалізація методу Sort()

Метод Sort реалізує сортування об'єктів у контейнері за властивістю Price. Використано алгоритм бульбашкового сортування. Для порівняння цін використовується допоміжний приватний метод GetPrice(object item), який за допомогою рефлексії отримує значення властивості "Price" об'єкта, якщо така властивість існує та має тип decimal. У випадку помилки під час отримання ціни, виводиться повідомлення про помилку в консоль.

5. Реалізація методу ToString()

Метод ToString перетворює вміст контейнера в рядок. Він ітерується по масиву items та викликає метод ToString() для кожного об'єкта, додаючи результат до результуючого рядка з символом нового рядка.

6. Консольна програма тестування

Консольна програма надає користувачу інтерактивне меню для тестування класу Container. Реалізовано режими автоматичної та ручної генерації об'єктів для додавання в контейнер, перегляд вмісту контейнера, сортування та видалення елементів за індексом. Для наочності виведення інформації використовується кольорове оформлення консолі. Для виведення вмісту контейнера реалізовано табличне представлення з пагінацією для зручного перегляду великої кількості об'єктів.

## Тестування коду

Тестування коду з автоматичним створенням відображено на рис. 1.

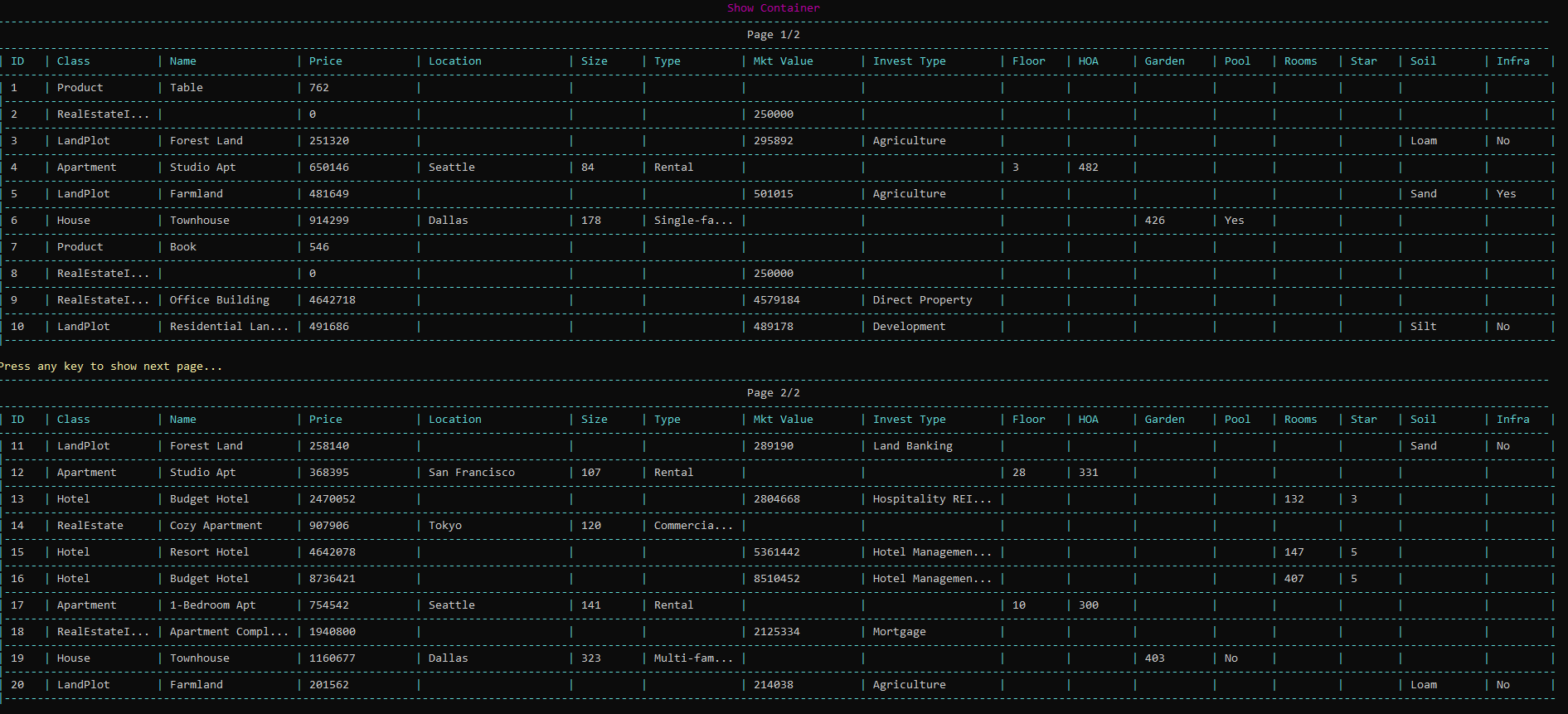


Рис. 1. Тестування коду з автоматичним створенням

Тестування коду сортування за цiною відображено на рис. 2.

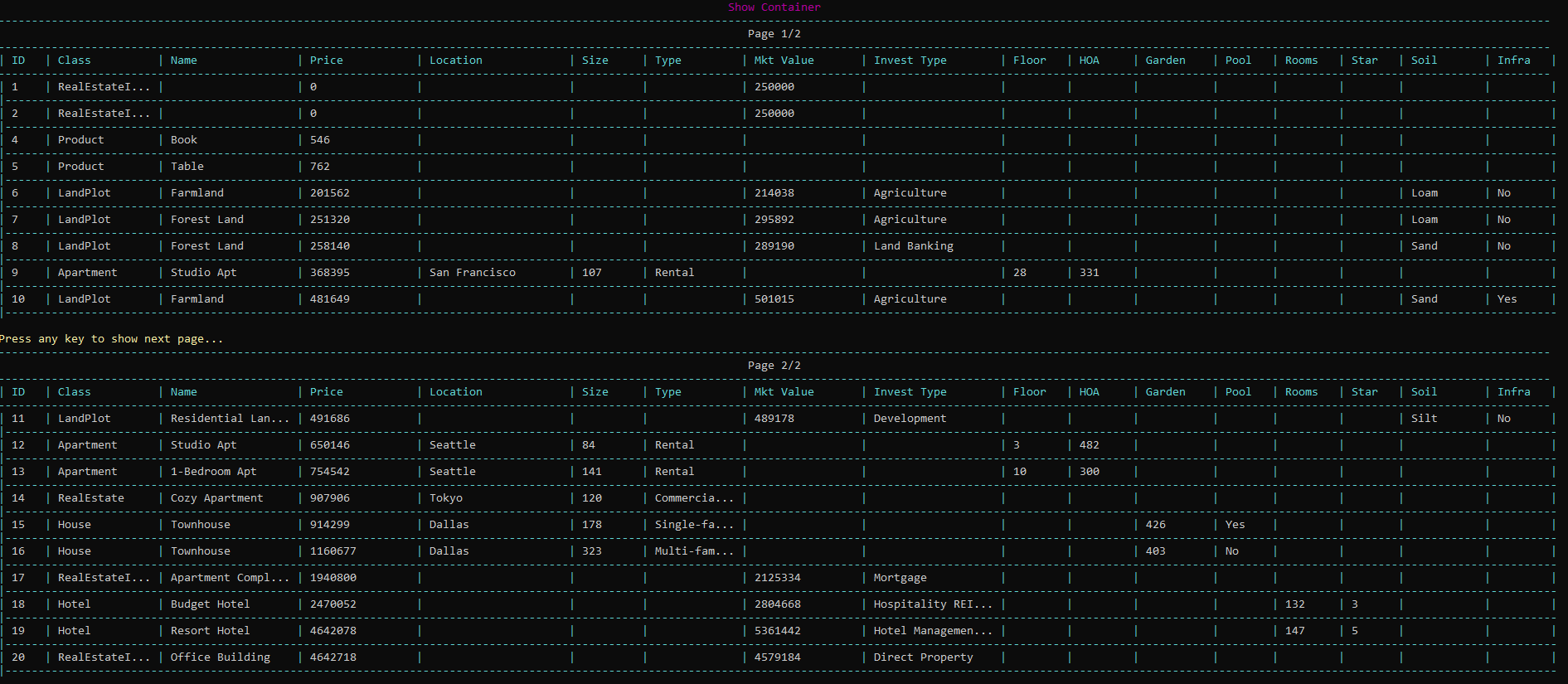


Рис. 2. Тестування коду сортування

Тестування коду видалення за iндексом відображено на рис. 3.

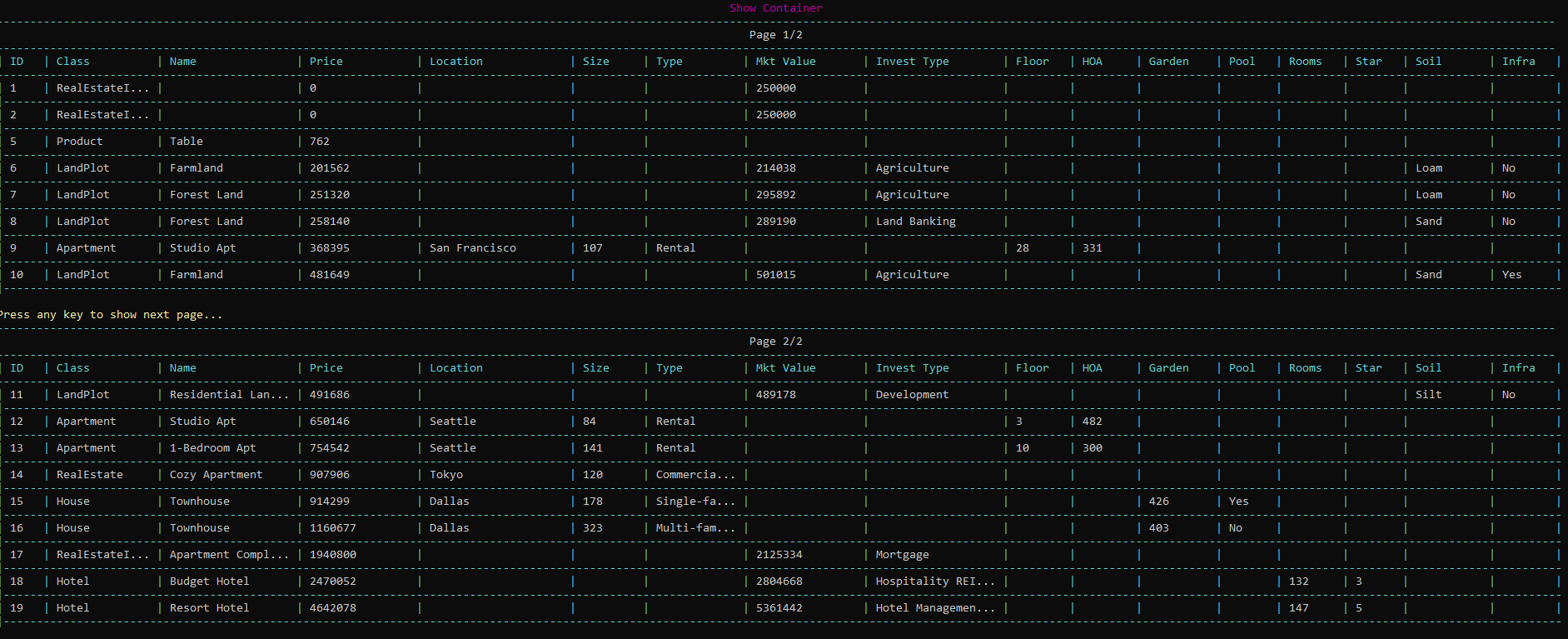


Рис. 3. Тестування коду видалення за iндексом

# Висновок

У ході виконання даної лабораторної роботи було успішно розроблено клас Container, який є фундаментальним елементом для організації зберігання та управління колекціями об'єктів у програмах. Створення цього класу, особливо з умовою ручної реалізації всіх основних операцій, стало цінним практичним досвідом, що дозволив глибше зрозуміти внутрішні механізми роботи з масивами та алгоритмами.

Реалізація методу Add продемонструвала важливість динамічного управління пам'яттю. Необхідність самостійно реалізувати розширення масиву при додаванні нових елементів дозволила на практиці усвідомити витрати на копіювання даних та вигоди від ефективного управління розміром масиву. Вибір стратегії подвоєння розміру масиву при розширенні, хоча і не єдиний можливий, є компромісом між частотою операцій розширення та витратами пам'яті.

Імплементація методу Sort з використанням алгоритму бульбашкового сортування стала практичним завданням на застосування базових алгоритмів сортування. Застосування рефлексії для отримання властивості Price та її значення продемонструвало гнучкість .NET Framework та можливість динамічно працювати з типами даних, хоча і з певними витратами на продуктивність.

Створення методу ToString для представлення вмісту контейнера у вигляді рядка підкреслило важливість надання користувачу зручного способу візуалізації даних. Реалізація табличного виведення з пагінацією в консольній програмі значно покращила зручність перегляду великих колекцій об'єктів, демонструючи важливість продуманого інтерфейсу користувача навіть у консольних застосунках.

# Додаток А

namespace lb\_4;

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)

{

if (price <= 0) throw new ValueLessThanZero("Price");

Name = name;

Price = price;

}

public override 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)

{

if (size <= 0) throw new ValueLessThanZero("Size");

Location = location;

Size = size;

Type = string.Empty;

}

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

{

if (size <= 0) throw new ValueLessThanZero("Size");

Location = location;

Size = size;

Type = type;

}

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

: base(name, price)

{

if (size <= 0) throw new ValueLessThanZero("Size");

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)

{

if (marketValue <= 0) throw new ValueLessThanZero("Market value");

Location = location;

MarketValue = marketValue;

InvestmentType = string.Empty;

}

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

{

if (marketValue <= 0) throw new ValueLessThanZero("Market value");

Location = location;

MarketValue = marketValue;

InvestmentType = investmentType;

}

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

: base(name, price)

{

if (marketValue <= 0) throw new ValueLessThanZero("Market value");

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)

{

if (floorNumber <= 0) throw new ValueLessThanZero("Floor number");

if (fees < 0) throw new ValueLessThanZero("Fee");

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)

{

if (floorNumber <= 0) throw new ValueLessThanZero("Floor number");

if (fees < 0) throw new ValueLessThanZero("Fee");

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)

{

if (GardenSize < 0) throw new ValueLessThanZero("Garden size");

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)

{

if (GardenSize < 0) throw new ValueLessThanZero("Garden size");

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 StarRating { get; set; }

public Hotel()

{

Rooms = 0;

StarRating = 0;

}

public Hotel(int rooms, int starRating)

{

if (rooms <= 0) throw new ValueLessThanZero("Rooms");

if (starRating <= 0 || starRating > 5) throw new ValueLessThanZero("Rating", "and not higher than 5");

Rooms = rooms;

StarRating = starRating;

}

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

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

{

if (rooms <= 0) throw new ValueLessThanZero("Rooms");

if (starRating <= 0 || starRating > 5) throw new ValueLessThanZero("Rating", "and not higher than 5");

Rooms = rooms;

StarRating = starRating;

}

public override string ToString()

{

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

}

}

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 ValueLessThanZero : Exception

{

public ValueLessThanZero(string name, string addition = "") : base(String.Format("{0} must be greater than zero {1}", name, addition)) { }

}

class Container

{

private Object[] items;

private int count;

private int size;

public Container()

{

items = new Product[1];

count = 0;

size = 1;

}

public void Add(object \_newObject)

{

if (count == size)

{

Object[] newArray = new Object[size \* 2];

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

{

newArray[i] = items[i];

}

items = newArray;

size \*= 2;

}

items[count] = \_newObject;

count++;

}

public object RemoveById(int \_index)

{

if (\_index > count)

throw new IndexOutOfRangeException();

if (count > 0 && \_index <= count)

{

object deletedObject = items[\_index];

Object[] newArray = new Object[size];

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

{

if (i == \_index)

continue;

newArray[i] = items[i];

}

items = newArray;

count--;

return deletedObject;

}

else

throw new IndexOutOfRangeException();

}

public void Sort()

{

try

{

for (int i = 0; i < count - 1; i++)

{

for (int j = 0; j < count – i - 1; j++)

{

if (GetPrice(items[j]) > GetPrice(items[j + 1]))

(items[j], items[j + 1]) = (items[j + 1], items[j]);

}

}

}

catch (Exception e)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(e.Message);

Console.ResetColor();

}

}

private decimal GetPrice(object item)

{

if (item != null)

{

var property = item.GetType().GetProperty("Price");

if (property != null && property.PropertyType == typeof(decimal))

{

return (decimal)property.GetValue(item);

}

}

return 0;

}

public string ToString()

{

string res = "";

foreach (var item in items)

{

if (item is null)

continue;

res += item.ToString() + "\n";

}

return res;

}

public Object[] GetItems()

{

return items;

}

public int GetCount()

{

return count;

}

}

class Program

{

static void Main()

{

Container container = new Container();

Random random = new Random();

while (true)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\n------ Menu ------");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("1. Automatic Generation");

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

Console.WriteLine("3. Show Container");

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("#. --- --- --- ---");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("4. Sort Container by Price");

Console.WriteLine("5. Remove Element by ID");

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("#. --- --- --- ---");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("q. Exit");

Console.ForegroundColor = ConsoleColor.Yellow;

Console.Write("Enter your choice: ");

Console.ResetColor();

string choice = Console.ReadLine();

switch (choice)

{

case "1":

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\n--- Automatic Generation ---");

Console.ResetColor();

Console.Write("Enter number of elements to generate: ");

if (int.TryParse(Console.ReadLine(), out int count))

{

AutomaticGeneration(container, random, count);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine($"Automatic generation of {count} elements complete.");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Invalid input for count. Generation cancelled.");

Console.ResetColor();

}

break;

case "2":

Console.ForegroundColor = ConsoleColor.Green;

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

Console.ResetColor();

ManualInput(container);

break;

case "3":

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\n--- Show Container ---");

Console.ResetColor();

ShowContainer(container);

break;

case "4":

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\n--- Sorted Container by Price ---");

Console.ResetColor();

container.Sort();

ShowContainer(container);

break;

case "5":

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\n--- Remove Element by Index ---");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("Enter element index to remove: ");

Console.ResetColor();

int index = Int32.Parse(Console.ReadLine()) - 1;

object deletedItem = container.RemoveById(index);

Console.ForegroundColor = ConsoleColor.DarkCyan;

Console.WriteLine($"Element '{deletedItem.ToString()}' was removed");

Console.ResetColor();

break;

case "q":

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Exiting...");

Console.ResetColor();

return;

default:

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Invalid choice. Please try again.");

Console.ResetColor();

break;

}

}

}

static void AutomaticGeneration(Container container, Random random, int count)

{

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

{

switch (random.Next(1, 9)) // Randomly choose a class

{

case 1: container.Add(GenerateRandomProduct(random)); break;

case 2: container.Add(GenerateRandomRealEstate(random)); break;

case 3: container.Add(GenerateRandomRealEstateInvestment(random)); break;

case 4: container.Add(GenerateRandomApartment(random)); break;

case 5: container.Add(GenerateRandomHouse(random)); break;

case 6: container.Add(GenerateRandomHotel(random)); break;

case 7: container.Add(GenerateRandomLandPlot(random)); break;

case 8: // Missing params generation

switch (random.Next(1, 7))

{

case 1: container.Add(new RealEstate("LocationA", 150.5)); break; // Missing Type

case 2: container.Add(new RealEstateInvestment("LocationB", 250000)); break; // Missing InvestmentType

case 3: container.Add(new Apartment(5, 150)); break; // Missing Name, Price, Location, Size, Type

case 4: container.Add(new House(500, true)); break; // Missing Name, Price, Location, Size, Type

case 5: container.Add(new Hotel(100, 4)); break; // Missing Name, Price, Location, MarketValue, InvestmentType

case 6: container.Add(new LandPlot("Chernozem", true)); break; // Missing Name, Price, Location, MarketValue, InvestmentType

}

break;

}

}

}

static void ManualInput(Container container)

{

Console.WriteLine("Choose class 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.ForegroundColor = ConsoleColor.Yellow;

Console.Write("Enter your choice: ");

Console.ResetColor();

string classChoice = Console.ReadLine();

try

{

switch (classChoice)

{

case "1":

container.Add(CreateManualProduct());

break;

case "2":

container.Add(CreateManualRealEstate());

break;

case "3":

container.Add(CreateManualRealEstateInvestment());

break;

case "4":

container.Add(CreateManualApartment());

break;

case "5":

container.Add(CreateManualHouse());

break;

case "6":

container.Add(CreateManualHotel());

break;

case "7":

container.Add(CreateManualLandPlot());

break;

default:

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Invalid class choice.");

Console.ResetColor();

break;

}

}

catch (ValueLessThanZero ex)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine($"Error: {ex.Message}");

Console.ResetColor();

}

catch (FormatException ex)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine($"Invalid input format: {ex.Message}");

Console.ResetColor();

}

}

static void ShowContainer(Container container)

{

Console.ForegroundColor = ConsoleColor.Magenta;

string title = "Show Container";

int tableWidth = CalculateTableWidth(); // Calculate total table width

if (container.GetCount() != 0)

Console.WriteLine(CenterString(title, tableWidth)); // Centered title

else

Console.WriteLine(title); // Centered title

Console.ForegroundColor = ConsoleColor.Cyan;

Console.ResetColor();

if (container.GetCount() == 0)

{

Console.WriteLine("Container is empty.");

return;

}

int itemsPerPage = 10; // Adjust as needed to fit your console

int pageCount = (int)Math.Ceiling((double)container.GetCount() / itemsPerPage);

for (int page = 0; page < pageCount; page++)

{

Console.ForegroundColor = ConsoleColor.Cyan;

DrawHorizontalLine(tableWidth); // Floor under title

string pageTitle = $"Page {page + 1}/{pageCount}";

Console.WriteLine(CenterString(pageTitle, tableWidth)); // Centered page number

DrawHorizontalLine(tableWidth);

WriteHeaderRow();

DrawHorizontalLine(tableWidth); // Separator after header

Console.ResetColor();

Object[] items = container.GetItems();

for (int i = page \* itemsPerPage; i < Math.Min((page + 1) \* itemsPerPage, container.GetCount()); i++)

{

var item = items[i];

if (item == null) continue;

WriteDataRow(i + 1, item);

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|"); // Add missing "|" at the end of row

DrawHorizontalLine(tableWidth); // Separator after each row

Console.ResetColor();

}

if (page + 1 < pageCount)

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("\nPress any key to show next page...");

Console.ReadKey();

Console.ResetColor();

}

}

}

// --- Helper methods for ShowContainer ---

static int CalculateTableWidth()

{

// Define column widths - Adjusted for better readability and alignment

const int idWidth = 4;

const int classWidth = 14;

const int nameWidth = 18;

const int priceWidth = 15;

const int locationWidth = 20;

const int sizeWidth = 8;

const int typeWidth = 12;

const int marketValueWidth = 15;

const int investmentTypeWidth = 18;

const int floorWidth = 7;

const int hoaWidth = 7;

const int gardenWidth = 9;

const int poolWidth = 6;

const int roomsWidth = 7;

const int starWidth = 6;

const int soilWidth = 10;

const int infraWidth = 7;

return idWidth + classWidth + nameWidth + priceWidth + locationWidth + sizeWidth + typeWidth + marketValueWidth + investmentTypeWidth + floorWidth + hoaWidth + gardenWidth + poolWidth + roomsWidth + starWidth + soilWidth + infraWidth + 51;

}

static void WriteHeaderRow()

{

// Define column widths - Adjusted for better readability and alignment

const int idWidth = 4;

const int classWidth = 14;

const int nameWidth = 18;

const int priceWidth = 15;

const int locationWidth = 20;

const int sizeWidth = 8;

const int typeWidth = 12;

const int marketValueWidth = 15;

const int investmentTypeWidth = 18;

const int floorWidth = 7;

const int hoaWidth = 7;

const int gardenWidth = 9;

const int poolWidth = 6;

const int roomsWidth = 7;

const int starWidth = 6;

const int soilWidth = 10;

const int infraWidth = 7;

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write($"| {"ID",-idWidth} | {"Class",-classWidth} | {"Name",-nameWidth} | {"Price",-priceWidth} | {"Location",-locationWidth} | {"Size",-sizeWidth} | {"Type",-typeWidth} | {"Mkt Value",-marketValueWidth} | {"Invest Type",-investmentTypeWidth} | {"Floor",-floorWidth} | {"HOA",-hoaWidth} | {"Garden",-gardenWidth} | {"Pool",-poolWidth} | {"Rooms",-roomsWidth} | {"Star",-starWidth} | {"Soil",-soilWidth} | {"Infra",-infraWidth} |\n");

Console.ResetColor();

}

static void WriteDataRow(int id, object item)

{

// Define column widths - Adjusted for better readability and alignment

const int idWidth = 4;

const int classWidth = 14;

const int nameWidth = 18;

const int priceWidth = 15;

const int locationWidth = 20;

const int sizeWidth = 8;

const int typeWidth = 12;

const int marketValueWidth = 15;

const int investmentTypeWidth = 18;

const int floorWidth = 7;

const int hoaWidth = 7;

const int gardenWidth = 9;

const int poolWidth = 6;

const int roomsWidth = 7;

const int starWidth = 6;

const int soilWidth = 10;

const int infraWidth = 7;

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {id,-idWidth} ");

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {Truncate(item.GetType().Name, classWidth - 3),-classWidth} "); // Class

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is Product product)

{

Console.Write($" {Truncate(product.Name, nameWidth - 3),-nameWidth} "); // Name

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {product.Price,-priceWidth} "); // Price

}

else

{

Console.Write($" {"",-nameWidth} "); // Name placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-priceWidth} "); // Price placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is RealEstate realEstate)

{

Console.Write($" {Truncate(realEstate.Location, locationWidth - 3),-locationWidth} "); // Location

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {realEstate.Size,-sizeWidth:F0} "); // Size

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {Truncate(realEstate.Type, typeWidth - 3),-typeWidth} "); // Type

}

else

{

Console.Write($" {"",-locationWidth} "); // Location placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-sizeWidth} "); // Size placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-typeWidth} "); // Type placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is RealEstateInvestment realEstateInvestment)

{

Console.Write($" {realEstateInvestment.MarketValue,-marketValueWidth} "); // Market Value

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {Truncate(realEstateInvestment.InvestmentType, investmentTypeWidth - 3),-investmentTypeWidth} "); // Investment Type

}

else

{

Console.Write($" {"",-marketValueWidth} "); // Market Value placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-investmentTypeWidth} "); // Investment Type placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is Apartment apartment)

{

Console.Write($" {apartment.FloorNumber,-floorWidth} "); // Floor

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {apartment.HOAFees,-hoaWidth} "); // HOA Fees

}

else

{

Console.Write($" {"",-floorWidth} "); // Floor placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-hoaWidth} "); // HOA Fees placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is House house)

{

Console.Write($" {house.GardenSize,-gardenWidth:F0} "); // Garden Size

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {(house.Pool ? "Yes" : "No"),-poolWidth} "); // Pool

}

else

{

Console.Write($" {"",-gardenWidth} "); // Garden Size placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-poolWidth} "); // Pool placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is Hotel hotel)

{

Console.Write($" {hotel.Rooms,-roomsWidth} "); // Rooms

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {hotel.StarRating,-starWidth} "); // Star Rating

}

else

{

Console.Write($" {"",-roomsWidth} "); // Rooms placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-starWidth} "); // Star Rating placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

if (item is LandPlot landPlot)

{

Console.Write($" {Truncate(landPlot.SoilType, soilWidth - 3),-soilWidth} "); // Soil Type

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {(landPlot.InfrastructureAccess ? "Yes" : "No"),-infraWidth} "); // Infrastructure Access

}

else

{

Console.Write($" {"",-soilWidth} "); // Soil Type placeholder

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|");

Console.ResetColor();

Console.Write($" {"",-infraWidth} "); // Infrastructure Access placeholder

}

Console.ForegroundColor = ConsoleColor.Cyan;

Console.Write("|"); // Add missing "|" at the end of row

Console.ResetColor();

Console.WriteLine();

}

static void DrawHorizontalLine(int tableWidth)

{

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine(new string('-', tableWidth));

Console.ResetColor();

}

static string CenterString(string s, int width)

{

int padding = (width - s.Length) / 2;

return new string(' ', padding) + s + new string(' ', width - s.Length - padding);

}

static string Truncate(string value, int maxLength)

{

if (string.IsNullOrEmpty(value)) return value;

return value.Length <= maxLength ? value : value.Substring(0, maxLength) + "...";

}

// --- Random Generators ---

static Product GenerateRandomProduct(Random random)

{

string[] names = { "Table", "Chair", "Lamp", "Phone", "Book" };

decimal price = random.Next(10, 1000);

return new Product(names[random.Next(names.Length)], price);

}

static RealEstate GenerateRandomRealEstate(Random random)

{

string[] names = { "Cozy Apartment", "Luxury Villa", "Small House", "Big Mansion" };

string[] locations = { "New York", "London", "Paris", "Tokyo", "Kyiv" };

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

decimal price = random.Next(100000, 1000000);

double size = random.Next(50, 500);

return new RealEstate(names[random.Next(names.Length)], price, locations[random.Next(locations.Length)], size, types[random.Next(types.Length)]);

}

static RealEstateInvestment GenerateRandomRealEstateInvestment(Random random)

{

string[] names = { "Office Building", "Shopping Mall", "Warehouse", "Apartment Complex" };

string[] locations = { "Chicago", "Los Angeles", "Houston", "Phoenix", "Philadelphia" };

string[] investmentTypes = { "REIT", "Direct Property", "Mortgage" };

decimal price = random.Next(500000, 5000000);

decimal marketValue = price + random.Next(-100000, 200000);

return new RealEstateInvestment(names[random.Next(names.Length)], price, locations[random.Next(locations.Length)], marketValue, investmentTypes[random.Next(investmentTypes.Length)]);

}

static Apartment GenerateRandomApartment(Random random)

{

string[] names = { "Studio Apt", "1-Bedroom Apt", "2-Bedroom Apt", "Penthouse" };

string[] locations = { "Miami", "San Francisco", "Seattle", "Boston", "Denver" };

string[] types = { "Condo", "Co-op", "Rental" };

decimal price = random.Next(200000, 800000);

double size = random.Next(40, 150);

int floorNumber = random.Next(1, 30);

decimal hoaFees = random.Next(100, 500);

return new Apartment(names[random.Next(names.Length)], price, locations[random.Next(locations.Length)], size, types[random.Next(types.Length)], floorNumber, hoaFees);

}

static House GenerateRandomHouse(Random random)

{

string[] names = { "Bungalow", "Townhouse", "Villa", "Cottage" };

string[] locations = { "Atlanta", "Dallas", "San Diego", "Orlando", "Las Vegas" };

string[] types = { "Single-family", "Multi-family" };

decimal price = random.Next(300000, 1200000);

double size = random.Next(100, 400);

double gardenSize = random.Next(0, 1000);

bool pool = random.Next(2) == 0;

return new House(names[random.Next(names.Length)], price, locations[random.Next(locations.Length)], size, types[random.Next(types.Length)], gardenSize, pool);

}

static Hotel GenerateRandomHotel(Random random)

{

string[] names = { "Luxury Hotel", "Budget Hotel", "Resort Hotel", "Boutique Hotel" };

string[] locations = { "Hawaii", "Bali", "Maldives", "Fiji", "Santorini" };

string[] investmentTypes = { "Hospitality REIT", "Hotel Management", "Timeshare" };

decimal price = random.Next(1000000, 10000000);

decimal marketValue = price + random.Next(-500000, 1000000);

int rooms = random.Next(50, 500);

int starRating = random.Next(3, 6);

return new Hotel(names[random.Next(names.Length)], price, locations[random.Next(locations.Length)], marketValue, investmentTypes[random.Next(investmentTypes.Length)], rooms, starRating);

}

static LandPlot GenerateRandomLandPlot(Random random)

{

string[] names = { "Farmland", "Forest Land", "Commercial Land", "Residential Land" };

string[] locations = { "Rural Area", "Suburban Area", "Urban Area", "Coastal Area" };

string[] investmentTypes = { "Land Banking", "Development", "Agriculture" };

string[] soilTypes = { "Loam", "Clay", "Sand", "Silt" };

decimal price = random.Next(50000, 500000);

decimal marketValue = price + random.Next(-20000, 50000);

bool infrastructureAccess = random.Next(2) == 0;

return new LandPlot(names[random.Next(names.Length)], price, locations[random.Next(locations.Length)], marketValue, investmentTypes[random.Next(investmentTypes.Length)], soilTypes[random.Next(soilTypes.Length)], infrastructureAccess);

}

// --- Manual Creation Methods ---

static Product CreateManualProduct()

{

Console.Write("Enter Product Name: ");

string name = Console.ReadLine();

Console.Write("Enter Product Price: ");

decimal price = decimal.Parse(Console.ReadLine());

return new Product(name, price);

}

static RealEstate CreateManualRealEstate()

{

Console.Write("Enter RealEstate Name: ");

string name = Console.ReadLine();

Console.Write("Enter RealEstate Price: ");

decimal price = decimal.Parse(Console.ReadLine());

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Console.Write("Enter Size: ");

double size = double.Parse(Console.ReadLine());

Console.Write("Enter Type: ");

string type = Console.ReadLine();

return new RealEstate(name, price, location, size, type);

}

static RealEstateInvestment CreateManualRealEstateInvestment()

{

Console.Write("Enter Investment Name: ");

string name = Console.ReadLine();

Console.Write("Enter Investment Price: ");

decimal price = decimal.Parse(Console.ReadLine());

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Console.Write("Enter Market Value: ");

decimal marketValue = decimal.Parse(Console.ReadLine());

Console.Write("Enter Investment Type: ");

string investmentType = Console.ReadLine();

return new RealEstateInvestment(name, price, location, marketValue, investmentType);

}

static Apartment CreateManualApartment()

{

Console.Write("Enter Apartment Name: ");

string name = Console.ReadLine();

Console.Write("Enter Apartment Price: ");

decimal price = decimal.Parse(Console.ReadLine());

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Console.Write("Enter Size: ");

double size = double.Parse(Console.ReadLine());

Console.Write("Enter Type: ");

string type = Console.ReadLine();

Console.Write("Enter Floor Number: ");

int floorNumber = int.Parse(Console.ReadLine());

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

decimal hoaFees = decimal.Parse(Console.ReadLine());

return new Apartment(name, price, location, size, type, floorNumber, hoaFees);

}

static House CreateManualHouse()

{

Console.Write("Enter House Name: ");

string name = Console.ReadLine();

Console.Write("Enter House Price: ");

decimal price = decimal.Parse(Console.ReadLine());

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Console.Write("Enter Size: ");

double size = double.Parse(Console.ReadLine());

Console.Write("Enter Type: ");

string type = Console.ReadLine();

Console.Write("Enter Garden Size: ");

double gardenSize = double.Parse(Console.ReadLine());

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

bool pool = bool.Parse(Console.ReadLine());

return new House(name, price, location, size, type, gardenSize, pool);

}

static Hotel CreateManualHotel()

{

Console.Write("Enter Hotel Name: ");

string name = Console.ReadLine();

Console.Write("Enter Hotel Price: ");

decimal price = decimal.Parse(Console.ReadLine());

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Console.Write("Enter Market Value: ");

decimal marketValue = decimal.Parse(Console.ReadLine());

Console.Write("Enter Investment Type: ");

string investmentType = Console.ReadLine();

Console.Write("Enter Number of Rooms: ");

int rooms = int.Parse(Console.ReadLine());

Console.Write("Enter Star Rating: ");

int starRating = int.Parse(Console.ReadLine());

return new Hotel(name, price, location, marketValue, investmentType, rooms, starRating);

}

static LandPlot CreateManualLandPlot()

{

Console.Write("Enter LandPlot Name: ");

string name = Console.ReadLine();

Console.Write("Enter LandPlot Price: ");

decimal price = decimal.Parse(Console.ReadLine());

Console.Write("Enter Location: ");

string location = Console.ReadLine();

Console.Write("Enter Market Value: ");

decimal marketValue = decimal.Parse(Console.ReadLine());

Console.Write("Enter Investment Type: ");

string investmentType = Console.ReadLine();

Console.Write("Enter Soil Type: ");

string soilType = Console.ReadLine();

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

bool infrastructureAccess = bool.Parse(Console.ReadLine());

return new LandPlot(name, price, location, marketValue, investmentType, soilType, infrastructureAccess);

}

}