**ПРАКТИЧНА РОБОТА №3**

**«КЛАСИ В C# ТА СПАДКУВАННЯ»**

**ВАРІАНТ №5**

**Мета роботи**: засвоїти основи побудови ієрархії класів на мові C#, набути практичних навичок використання прийомів об’єктно-орієнтованого програмування.

**Вправа 3:**

3.1. Перелік класів: Організація, Страхова компанія, Суднобудівельна компанія, Завод.

Повний лістинг програми:

using System;

using System.Collections.Generic;

namespace prac3\_console

{

public abstract class Organization

{

private string \_name;

private string \_industry;

private string \_area\_served;

public static LinkedList<Organization> List = new LinkedList<Organization>();

public static void CheckList()

{

foreach (var item in List)

{

item.GetInfo();

}

}

public string Name

{

get { return \_name; }

set { \_name = value; }

}

public string Industry

{

get { return \_industry; }

set { \_industry = value; }

}

public string Area\_served

{

get { return \_area\_served; }

set { \_area\_served = value; }

}

public abstract void GetInfo();

public void Add()

{

List.AddLast(this);

}

public Organization()

{

\_name = "No name";

\_industry = null;

\_area\_served = null;

}

public Organization(string name, string industry, string area\_served)

{

\_name = name;

\_industry = industry;

\_area\_served = area\_served;

}

~Organization()

{

Console.WriteLine("Desposing of" + \_name);

}

}

public class Factory : Organization

{

private string \_head;

private int \_number\_of\_employees;

private string \_city;

public string Head

{

get { return \_head; }

set { \_head = value; }

}

public int N\_O\_E

{

get { return \_number\_of\_employees; }

set { \_number\_of\_employees = value; }

}

public string City

{

get { return \_city; }

set { \_city = value; }

}

public override void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_head, \_number\_of\_employees, \_city);

}

public Factory() : base()

{

\_head = null;

\_number\_of\_employees = 0;

\_city = null;

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

\_head = head;

\_city = city;

\_number\_of\_employees = number\_of\_employees;

}

~Factory() { }

}

public class Insurance : Organization

{

private int \_client\_number;

private string \_speciality;

public int CN

{

get { return \_client\_number; }

set { \_client\_number = value; }

}

public string Speciality

{

get { return \_speciality; }

set { \_speciality = value; }

}

public override void GetInfo()

{

Console.WriteLine($"Insurance: \n\tName of organisation: {Name}");

Console.WriteLine("\tNumber of clients: {0}\n\tSpecialization: {1}\n", \_client\_number, \_speciality);

}

public Insurance() : base() { }

public Insurance(string name, string industry, string area\_served,

int client\_number, string spec) : base(name, industry, area\_served)

{

this.\_client\_number = client\_number;

this.\_speciality = spec;

}

~Insurance() { }

}

public class BuildingCompany : Organization

{

private string \_speciality;

private string \_city;

private int \_number\_of\_employees;

public string Speciality { set => \_speciality = value; get => \_speciality; }

public string City { set => \_city = value; get => \_city; }

public int N\_O\_E { set => \_number\_of\_employees = value; get => \_number\_of\_employees; }

public override void GetInfo()

{

Console.WriteLine($"Building Company: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tSpeciality: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_speciality, \_number\_of\_employees, \_city);

}

public BuildingCompany() { }

public BuildingCompany(string name, string industry, string area\_wide, string spec, string city, int n\_o\_e) : base(name, industry, area\_wide)

{

}

~BuildingCompany() { }

}

internal class Program

{

static void Main(string[] args)

{

Factory f = new Factory("agrr", "oil", "Country", "Popolski", "Kyiv", 900);

f.GetInfo();

BuildingCompany bc = new BuildingCompany("Building Co.", "skys", "City", "Gladkevich", "Odesa", 1900);

Insurance ins = new Insurance("Insurance Co. ", "Money", "City",300, "Cars");

Organization.CheckList();

f.Add();

bc.Add();

ins.Add();

Organization.CheckList();

}

}

}

Ієрархія класів, визначена у програмі:

Клас-батько: *Organiztion*

Класи-спадкоємці від *Organization*: *BuildingCompany, Factory, Insurance.*

3.2. Опис абстрактного класу *Organiztion:*

public abstract class Organization

{

private string \_name;

private string \_industry;

private string \_area\_served;

public static LinkedList<Organization> List = new LinkedList<Organization>();

public static void CheckList()

{

foreach (var item in List)

{

item.GetInfo();

}

}

public string Name

{

get { return \_name; }

set { \_name = value; }

}

public string Industry

{

get { return \_industry; }

set { \_industry = value; }

}

public string Area\_served

{

get { return \_area\_served; }

set { \_area\_served = value; }

}

public abstract void GetInfo();

public void Add()

{

List.AddLast(this);

}

public Organization()

{

\_name = "No name";

\_industry = null;

\_area\_served = null;

}

public Organization(string name, string industry, string area\_served)

{

\_name = name;

\_industry = industry;

\_area\_served = area\_served;

}

~Organization()

{

Console.WriteLine("Desposing of" + \_name);

}

}

В класі визначені 2 конструктори, перший пустий, в другий передаються такі дані, як Назва, Індустрія та Область займаності компанії. Також визначений деструктор:

public Organization()

{

\_name = "No name";

\_industry = null;

\_area\_served = null;

}

public Organization(string name, string industry, string area\_served)

{

\_name = name;

\_industry = industry;

\_area\_served = area\_served;

}

~Organization()

{

Console.WriteLine("Desposing of" + \_name);

}

Визначаються декілька приватних змінних, проводиться їх інкапсуляція:

private string \_name;

private string \_industry;

private string \_area\_served;

public string Name

{

get { return \_name; }

set { \_name = value; }

}

public string Industry

{

get { return \_industry; }

set { \_industry = value; }

}

public string Area\_served

{

get { return \_area\_served; }

set { \_area\_served = value; }

}

Створення об’єкту колекції *LinkedList* на основі класу *Organization* та метод CheckList , що виводить всі елементи списку:

public static LinkedList<Organization> List = new LinkedList<Organization>();

public static void CheckList()

{

foreach (var item in List)

{

item.GetInfo();

}

}

LinkedList представляє собою однозв’язний список в якому кожен елемент зберігає посилання на наступний об’єкт списку. Схема роботи списку показана на рисунку 1.

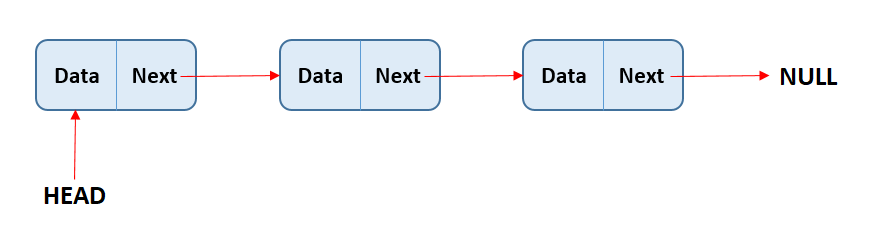


Рисунок 1 – Схема роботи однозв’язного списку

Метод *Add* виконує функцію додавання елементів у кінець двозв’язанного списку *List*:

public void Add()

{

List.AddLast(this);

}

3.3. Опис класу-спадкоємцю *Factory:*

public class Factory : Organization

{

private string \_head;

private int \_number\_of\_employees;

private string \_city;

public string Head

{

get { return \_head; }

set { \_head = value; }

}

public int N\_O\_E

{

get { return \_number\_of\_employees; }

set { \_number\_of\_employees = value; }

}

public string City

{

get { return \_city; }

set { \_city = value; }

}

public override void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_head, \_number\_of\_employees, \_city);

}

public Factory() : base()

{

\_head = null;

\_number\_of\_employees = 0;

\_city = null;

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

\_head = head;

\_city = city;

\_number\_of\_employees = number\_of\_employees;

}

~Factory() { }

}

Визначення приватних змінних та присвоєння їм властивостей:

private string \_head;

private int \_number\_of\_employees;

private string \_city;

public string Head

{

get { return \_head; }

set { \_head = value; }

}

public int N\_O\_E

{

get { return \_number\_of\_employees; }

set { \_number\_of\_employees = value; }

}

public string City

{

get { return \_city; }

set { \_city = value; }

}

Опис двох конструкторів та деструктора класу *Factory*. Перший конструктор нічого не приймає, в другий передаються змінні string name, string industry, string area\_served, string head, string city, int number\_of\_employees серед яких name, industry, та area\_served наслідуються від базового класу яким є *Organization.*

public Factory() : base()

{

\_head = null;

\_number\_of\_employees = 0;

\_city = null;

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

\_head = head;

\_city = city;

\_number\_of\_employees = number\_of\_employees;

}

~Factory() { }

}

Метод GetInfo виводить у консоль інформацію про поточну організацію:

public override void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_head, \_number\_of\_employees, \_city);

}

Класи *BuildingCompany, Insurance* мають таку ж реалізацію як і клас *Factory.*



Рисунок 3.2 - UML-діаграма.

3.5. Демонстрація роботи проекту:

У статичному методі *Main* ініціалізуються екземпляри класів-спадкоємців. За допомогою методу *Add* класу *Organization* екземпляри класів-спадкоємців додаються до списку *List*(Рисунок 3.2).

static void Main(string[] args)

{

Factory f = new Factory("agrr", "oil", "Country", "Popolski", "Kyiv", 900);

f.GetInfo();

BuildingCompany bc = new BuildingCompany("Building Co.", "skys", "City", "Gladkevich", "Odesa", 1900);

Insurance ins = new Insurance("Insurance Co. ", "Money", "City",300, "Cars");

Organization.CheckList();

f.Add();

bc.Add();

ins.Add();

Organization.CheckList();

}

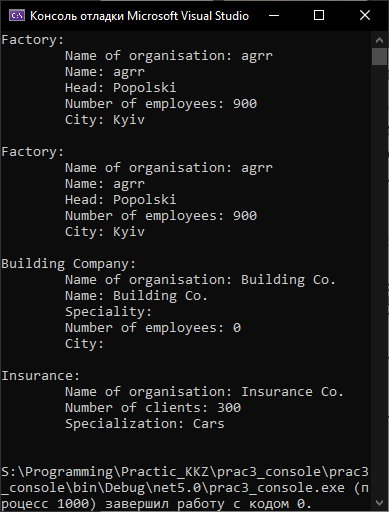


Рисунок 3.3 – результат роботи програми.

Вправа 1.

Лістинг до вправи:

using System;

using System.Threading;

class Car

{

private String mName;

public Car()

{

mName = "No Name";

}

public Car(String name)

{

mName = name;

}

public void Print()

{

Console.WriteLine("Name : " + mName);

}

~Car()

{

Console.WriteLine("Disposing of " + mName);

}

}

class TestMain

{

public static void Main()

{

for (int i = 0; i < 9000; ++i)

{

Car a = new Car("Name:" + i);

a.Print();

}

}

}

В ході виконання програми у циклі створюється велика кількість екземплярів класу та виводяться на екран їх імена далі після їх створення повинні виводитися повідомлення від деструктора, але виходячи з правил за якими працює Garbage Collector программа завершує свою роботу раніше чим встигнуть вивестися повідомлення деструктора.

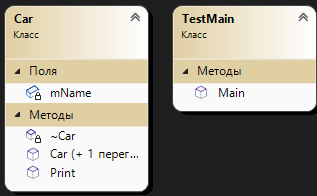


Рисунок 4 – UML діаграма класів

Вправа 2.1.

Повний лістинг:

using System;

public class Base

{

public void PrintItem()

{

for (int i = 0; i < 10; ++i)

Console.WriteLine("Item: " + this.GetItem(i));

}

public virtual int GetItem(int index)

{

return 0;

}

}

public class Derived : Base

{

public override int GetItem(int index)

{

return index \* index;

}

}

public class TestMain

{

public static void Main()

{

new Derived().PrintItem();

}

}

Напочатку створений клас Base в якому створені 2 метода який виводить результат роботи віртуального методу GetItem. У класі Derived, який успадковується від класу Base, далі перевизначається метод GetItem. У методі Main створюється об’єкт Derived та ініціалізується метод PrintItem. Результат роботи програми показан на рисунку 5.

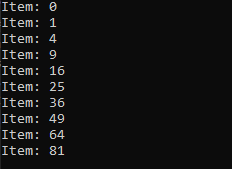


Рисунок 5 – Результат виконання програми

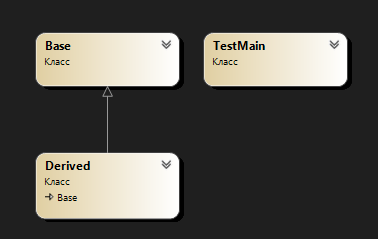


Рисунок 6 – UML діаграма класів

Вправа 2.2.

У цій вправі єдиною різницею з попередньою є в тому що метод GetItem є абстрактним. А тому в базовому калсі його тіло не створено, тому він має бути обов’язково перевизначений у похідному.

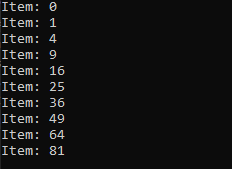


Рисунок 7 – Результат виконання програми

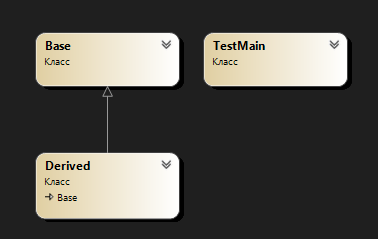


Рисунок 8 – UML діаграма класів