Лабораторна робота № 1

Тема: Принципи програмування. DRY, KISS, SOLID, YAGNI та ін.

Мета роботи: навчитися дотримуватися принципів програмування та обґрунтовувати їх.

Хід роботи

Завдання №1-2. Виконати завдання з дотриманням відомих Вам принципів програмування.

1. Створіть систему класів для обліку зоопарку. Ви можете створювати класи для різних видів і підвидів тварин; для вольєрів різних розмірів і типів; корму для тварин; працівників зоопарку.

2. Створіть класи інвентаризації, для виведення на екран інформації про наявних тварин, кількості співробітників тощо.

Лістинг Program.cs:

```
using System;
using System.Text;
using ZooClassLibrary;

namespace ZooManagementSystem
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.OutputEncoding = Encoding.UTF8;

            Zoo zoo = new Zoo();
            InitializeZoo(zoo);

            while (true)
            {
                Console.Clear();
                Console.WriteLine("Choose an action:");
                Console.WriteLine("1 - Action with animals");
                Console.WriteLine("2 - Action with enclosures");
                Console.WriteLine("3 - Action with employees");
                Console.WriteLine("4 - Action with food");
                Console.WriteLine("5 - View All");
                Console.WriteLine("Exit");

                string choice = Console.ReadLine();
                switch (choice)
                {
                    case "1":
                        AnimalActions(zoo);
                        break;
                    case "2":
                        EnclosureActions(zoo);
                        break;
                    case "3":
                        EmployeeActions(zoo);
                        break;
                    case "4":
```

```
                FoodActions(zoo);
                break;
            case "5":
                ViewAll(zoo);
                break;
            case "Exit":
            case "exit":
                Environment.Exit(0);
                break;
            default:
                Console.WriteLine("Invalid selection. Please try again.");
                break;
        }
    }
}

//Standard data
static void InitializeZoo(Zoo zoo)
{
    zoo.Animals.Add(new Animal { Species = "Lion", Subspecies = "African", Age = 5 });
    zoo.Animals.Add(new Animal { Species = "Tiger", Subspecies = "Bengal", Age = 4 });
    zoo.Animals.Add(new Animal { Species = "Elephant", Subspecies = "Indian", Age = 10 });

    zoo.Enclosures.Add(new Enclosure { Id = 1, Type = "Medium 1", Area = 100 });
    zoo.Enclosures.Add(new Enclosure { Id = 2, Type = "Medium 2", Area = 80 });
    zoo.Enclosures.Add(new Enclosure { Id = 3, Type = "Large 1", Area = 200 });

    zoo.Employees.Add(new Employee { Name = "Ivan", Age = 35, Position = "Veterinarian" });
    zoo.Employees.Add(new Employee { Name = "Maria", Age = 28, Position = "Maintenance" });
    zoo.Employees.Add(new Employee { Name = "Petro", Age = 40, Position = "Guardian" });

    zoo.Foods.Add(new Food { Name = "Meat", Quantity = 100 });
    zoo.Foods.Add(new Food { Name = "Vegetables", Quantity = 200 });
    zoo.Foods.Add(new Food { Name = "Fruit", Quantity = 150 });
}

//Actions with animals
static void AnimalActions(Zoo zoo)
{
    Console.OutputEncoding = Encoding.UTF8;


    while (true)
    {
        Console.Clear();
        DisplayAnimalMenu();

        string choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                ViewAllAnimals(zoo);
                break;
            case "2":
                ChangeAnimalData(zoo);
                break;
            case "3":
                DeleteAnimal(zoo);
                break;
            case "4":
                AddAnimal(zoo);
                break;
            case "5":
                return;
            default:
                Console.WriteLine("Invalid selection. Please try again.");
                break;
        }
        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine();
    }
}
static void DisplayAnimalMenu()
{
    Console.WriteLine("Action with animals:");
    Console.WriteLine("1 - View all animals");
    Console.WriteLine("2 - Change animal data");
```

```
    Console.WriteLine("3 - Delete animal");
    Console.WriteLine("4 - Add an animal");
    Console.WriteLine("5 - Return to main menu");
}
static void ViewAllAnimals(Zoo zoo)
{
    Console.WriteLine("List of all animals:");
    foreach (var animal in zoo.Animals)
    {
        Console.WriteLine($"Species: {animal.Species}, Subspecies: {animal.Subspecies}, Age: {animal.Age}");
    }
}
static void ChangeAnimalData(Zoo zoo)
{
    Console.WriteLine("Enter the type of animal to change data:");
    string speciesToEdit = Console.ReadLine();
    Console.WriteLine("Enter the animal subspecies to change data:");
    string subspeciesToEdit = Console.ReadLine();
    Animal animalToEdit = zoo.Animals.Find(a => a.Species == speciesToEdit && a.Subspecies == subspeciesToEdit);
    if (animalToEdit != null)
    {
        Console.WriteLine($"Enter new age for animal {speciesToEdit} {subspeciesToEdit}:");
        if (int.TryParse(Console.ReadLine(), out int newAnimalAge))
        {
            animalToEdit.Age = newAnimalAge;
            Console.WriteLine("Data changed successfully.");
        }
        else
        {
            Console.WriteLine("Invalid age format. Please enter an integer.");
        }
    }
    else
    {
        Console.WriteLine("No animal found.");
    }
}
static void DeleteAnimal(Zoo zoo)
{
    Console.WriteLine("Enter the type of animal to delete:");
    string speciesToDelete = Console.ReadLine();
    Console.WriteLine("Enter the animal subspecies to delete:");
    string subspeciesToDelete = Console.ReadLine();
    Animal animalToDelete = zoo.Animals.Find(a => a.Species == speciesToDelete && a.Subspecies == subspeciesToDelete);
    if (animalToDelete != null)
    {
        zoo.Animals.Remove(animalToDelete);
        Console.WriteLine($"Animal {speciesToDelete} {subspeciesToDelete} has been removed from the zoo.");
    }
    else
    {
        Console.WriteLine("No animal found.");
    }
}
static void AddAnimal(Zoo zoo)
{
    Console.WriteLine("Enter the appearance of the new animal:");
    string newSpecies = Console.ReadLine();
    if (!ValidateADDInput(newSpecies, "Invalid input. Appearance cannot be empty."))
        return;

    Console.WriteLine("Enter the subspecies of the new animal:");
    string newSubspecies = Console.ReadLine();
    if (!ValidateADDInput(newSubspecies, "Invalid input. Subspecies cannot be empty."))
        return;

    Console.WriteLine("Enter the age of the new animal:");
    string ageInput = Console.ReadLine();
    int newAge;
    if (!ValidateADDIntInput(ageInput, out newAge, "Invalid input. Age must be a valid integer."))
        return;

    zoo.Animals.Add(new Animal { Species = newSpecies, Subspecies = newSubspecies, Age = newAge });
    Console.WriteLine($"Animal {newSpecies} {newSubspecies} has been added to the zoo.");
}

//Actions with enclosures
static void EnclosureActions(Zoo zoo)
```

```
{
    Console.OutputEncoding = Encoding.UTF8;
    while (true)
    {
        Console.Clear();
        DisplayEnclosureMenu();

        string choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                ViewAllEnclosures(zoo);
                break;
            case "2":
                ChangeEnclosureData(zoo);
                break;
            case "3":
                DeleteEnclosure(zoo);
                break;
            case "4":
                AddEnclosure(zoo);
                break;
            case "5":
                return;
            default:
                Console.WriteLine("Invalid selection. Please try again.");
                break;
        }

        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine();
    }
}
static void DisplayEnclosureMenu()
{
    Console.WriteLine("Action with enclosures:");
    Console.WriteLine("1 - View all enclosures");
    Console.WriteLine("2 - Change data about the enclosure");
    Console.WriteLine("3 - Delete enclosure");
    Console.WriteLine("4 - Add enclosure");
    Console.WriteLine("5 - Return to main menu");
}
static void ViewAllEnclosures(Zoo zoo)
{
    Console.WriteLine("List of all enclosures:");
    foreach (var enclosure in zoo.Enclosures)
    {
        Console.WriteLine($"Id: {enclosure.Id}, Type: {enclosure.Type}, Area: {enclosure.Area}");
    }
}
static void ChangeEnclosureData(Zoo zoo)
{
    Console.WriteLine("Enter enclosure ID to change data:");
    int enclosureIdToEdit = int.Parse(Console.ReadLine());
    Enclosure enclosureToEdit = zoo.Enclosures.Find(e => e.Id == enclosureIdToEdit);
    if (enclosureToEdit != null)
    {
        Console.WriteLine($"Enter a new type for enclosure with ID {enclosureIdToEdit}:");
        string newType = Console.ReadLine();
        Console.WriteLine($"Enter a new area for the enclosure with the ID {enclosureIdToEdit}:");
        double newArea = double.Parse(Console.ReadLine());
        enclosureToEdit.Type = newType;
        enclosureToEdit.Area = newArea;
        Console.WriteLine("Data changed successfully.");
    }
    else
    {
        Console.WriteLine("Enclosure not found");
    }
}
static void DeleteEnclosure(Zoo zoo)
{
    Console.WriteLine("Enter the enclosure ID to delete:");
    int enclosureIdToDelete = int.Parse(Console.ReadLine());
    Enclosure enclosureToDelete = zoo.Enclosures.Find(e => e.Id == enclosureIdToDelete);
    if (enclosureToDelete != null)
    {
```

```
            zoo.Enclosures.Remove(enclosureToDelete);
            Console.WriteLine($"The enclosure with ID {enclosureIdToDelete} has been deleted from the zoo.");
        }
        else
        {
            Console.WriteLine("Enclosure not found");
        }
    }
    static void AddEnclosure(Zoo zoo)
    {
        int newEnclosureId;
        Console.WriteLine("Enter the ID of the new enclosure:");
        string idInput = Console.ReadLine();
        if (!ValidateADDIntInput(idInput, out newEnclosureId, "Invalid input. Enclosure ID must be a valid integer."))
            return;

        Console.WriteLine("Enter the type of new enclosure:");
        string newEnclosureType = Console.ReadLine();
        if (!ValidateADDInput(newEnclosureType, "Invalid input. Enclosure type cannot be empty."))
            return;

        Console.WriteLine("Enter the area of the new enclosure:");
        string areaInput = Console.ReadLine();
        double newEnclosureArea;
        if (!double.TryParse(areaInput, out newEnclosureArea))
        {
            Console.WriteLine("Invalid input. Enclosure area must be a valid number.");
            return;
        }

        zoo.Enclosures.Add(new Enclosure { Id = newEnclosureId, Type = newEnclosureType, Area = newEnclosureArea });
        Console.WriteLine($"The enclosure with ID {newEnclosureId} has been added to the zoo.");
    }

    //Actions with employees
    static void EmployeeActions(Zoo zoo)
    {
        Console.OutputEncoding = Encoding.UTF8;
        while (true)
        {
            Console.Clear();
            DisplayEmployeeMenu();

            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    ViewAllEmployees(zoo);
                    break;
                case "2":
                    ChangeEmployeeData(zoo);
                    break;
                case "3":
                    DeleteEmployee(zoo);
                    break;
                case "4":
                    AddEmployee(zoo);
                    break;
                case "5":
                    return;
                default:
                    Console.WriteLine("Invalid selection. Please try again.");
                    break;
            }

            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
    }
    static void DisplayEmployeeMenu()
    {
        Console.WriteLine("Action with employees:");
        Console.WriteLine("1 - View all employees");
        Console.WriteLine("2 - Change employee data");
        Console.WriteLine("3 - Delete employee");
        Console.WriteLine("4 - Add an employee");
        Console.WriteLine("5 - Return to main menu");
```

```
}
static void ViewAllEmployees(Zoo zoo)
{
    Console.WriteLine("List of all workers:");
    foreach (var employee in zoo.Employees)
    {
        Console.WriteLine($"Name: {employee.Name}, Age: {employee.Age}, Position: {employee.Position}");
    }
}
static void ChangeEmployeeData(Zoo zoo)
{
    Console.WriteLine("Enter the name of the employee to change the data:");
    string nameToEdit = Console.ReadLine();
    Employee employeeToEdit = zoo.Employees.Find(e => e.Name == nameToEdit);
    if (employeeToEdit != null)
    {
        Console.WriteLine($"Enter new age for employee {nameToEdit}:");
        if (int.TryParse(Console.ReadLine(), out int newAge))
        {
            employeeToEdit.Age = newAge;
            Console.WriteLine("Data changed successfully.");
        }
        else
        {
            Console.WriteLine("Invalid age format. Please enter an integer.");
        }
    }
    else
    {
        Console.WriteLine("Employee not found.");
    }
}
static void DeleteEmployee(Zoo zoo)
{
    Console.WriteLine("Enter the name of the employee to delete:");
    string nameToDelete = Console.ReadLine();
    Employee employeeToDelete = zoo.Employees.Find(e => e.Name == nameToDelete);
    if (employeeToDelete != null)
    {
        zoo.Employees.Remove(employeeToDelete);
        Console.WriteLine($"Employee {nameToDelete} has been removed from the zoo.");
    }
    else
    {
        Console.WriteLine("Employee not found.");
    }
}
static void AddEmployee(Zoo zoo)
{
    Console.WriteLine("Enter the name of the new employee:");
    string newName = Console.ReadLine();
    if (!ValidateADDInput(newName, "Invalid input. Employee name cannot be empty."))
        return;

    Console.WriteLine($"Enter age for employee {newName}:");
    string ageInput = Console.ReadLine();
    int newEmployeeAge;
    if (!ValidateADDIntInput(ageInput, out newEmployeeAge, "Invalid input. Age must be a valid integer."))
        return;

    Console.WriteLine($"Enter the job title for employee {newName}:");
    string newPosition = Console.ReadLine();
    if (!ValidateADDInput(newPosition, "Invalid input. Job title cannot be empty."))
        return;

    zoo.Employees.Add(new Employee { Name = newName, Age = newEmployeeAge, Position = newPosition });
    Console.WriteLine($"Employee {newName} has been added to the zoo.");
}

//Actions with foods
static void FoodActions(Zoo zoo)
{
    Console.OutputEncoding = Encoding.UTF8;
    while (true)
    {
        Console.Clear();
        DisplayFoodMenu();
```

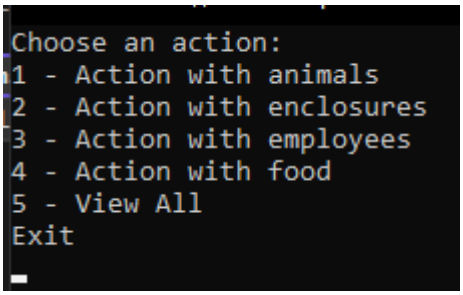
```
        string choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                ViewAvailableFood(zoo);
                break;
            case "2":
                AddFoodType(zoo);
                break;
            case "3":
                DeleteFoodType(zoo);
                break;
            case "4":
                return;
            default:
                Console.WriteLine("Invalid selection. Please try again.");
                break;
        }

        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine();
    }
}
static void DisplayFoodMenu()
{
    Console.WriteLine("Action with food:");
    Console.WriteLine("1 - View the list of available food");
    Console.WriteLine("2 - Add a new type of food");
    Console.WriteLine("3 - Delete type of food");
    Console.WriteLine("4 - Return to main menu");
}
static void ViewAvailableFood(Zoo zoo)
{
    Console.WriteLine("List of available food:");
    foreach (var food in zoo.Foods)
    {
        Console.WriteLine($"Name: {food.Name}, Quantity: {food.Quantity}");
    }
}
static void DeleteFoodType(Zoo zoo)
{
    Console.WriteLine("Enter the name of the type of food to delete:");
    string foodToDelete = Console.ReadLine();
    Food foodItemToDelete = zoo.Foods.Find(f => f.Name == foodToDelete);
    if (foodItemToDelete != null)
    {
        zoo.Foods.Remove(foodItemToDelete);
        Console.WriteLine($"Food type {foodToDelete} deleted.");
    }
    else
    {
        Console.WriteLine($"Food type {foodToDelete} not found.");
    }
}
static void AddFoodType(Zoo zoo)
{
    Console.WriteLine("Enter the name of the new type of food:");
    string newFoodName = Console.ReadLine();
    if (!ValidateADDInput(newFoodName, "Invalid input. Food name cannot be empty."))
        return;

    Console.WriteLine("Enter the amount of food:");
    string quantityInput = Console.ReadLine();
    int newFoodQuantity;
    if (!ValidateADDIntInput(quantityInput, out newFoodQuantity, "Invalid input. Quantity must be a valid integer."))
        return;

    zoo.Foods.Add(new Food { Name = newFoodName, Quantity = newFoodQuantity });
    Console.WriteLine($"Food type {newFoodName} added.");
}

//Display of everything
static void ViewAll(Zoo zoo)
{
    Console.OutputEncoding = Encoding.UTF8;
    Console.Clear();
    Console.WriteLine("Zoo Information:");
```

```
            Console.WriteLine("Animals:");
            ViewAllAnimals(zoo);

            Console.WriteLine("\nAviaries:");
            ViewAllEnclosures(zoo);

            Console.WriteLine("\nEmployees:");
            ViewAllEmployees(zoo);

            Console.WriteLine("\nPress Enter to continue...");
            Console.ReadLine();
        }

        //Common methods for input validation
        static bool ValidateADDInput(string input, string errorMessage)
        {
            if (string.IsNullOrWhiteSpace(input))
            {
                Console.WriteLine(errorMessage);
                return false;
            }
            return true;
        }
        static bool ValidateADDIntInput(string input, out int result, string errorMessage)
        {
            if (!int.TryParse(input, out result))
            {
                Console.WriteLine(errorMessage);
                return false;
            }
            return true;
        }
    }
}
```

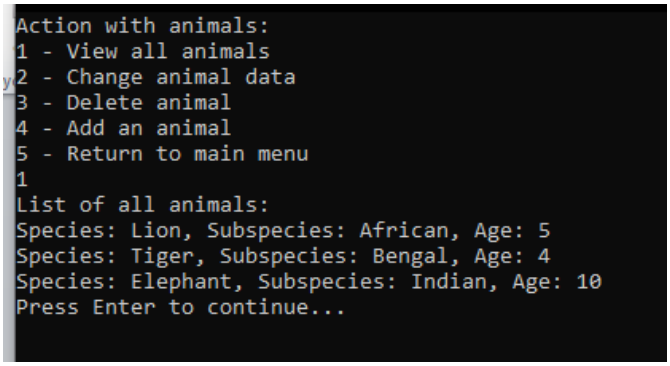
Лістинг Classes.cs:

```
using System.Collections.Generic;

namespace ZooClassLibrary
{
    public class Animal
    {
        public string Species { get; set; }
        public string Subspecies { get; set; }
        public int Age { get; set; }
    }
    public class Enclosure
    {
        public int Id { get; set; }
        public string Type { get; set; }
        public double Area { get; set; }
        public List<Animal> Animals { get; set; } = new List<Animal>();
    }
    public class Employee
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public string Position { get; set; }
    }
    public class Food
    {
        public string Name { get; set; }
        public int Quantity { get; set; }
    }
    public class Zoo
    {
        public List<Food> Foods { get; set; }
        public List<Animal> Animals { get; set; }
        public List<Enclosure> Enclosures { get; set; }
        public List<Employee> Employees { get; set; }

        public Zoo()
        {
            Foods = new List<Food>();
            Animals = new List<Animal>();
            Enclosures = new List<Enclosure>();
```

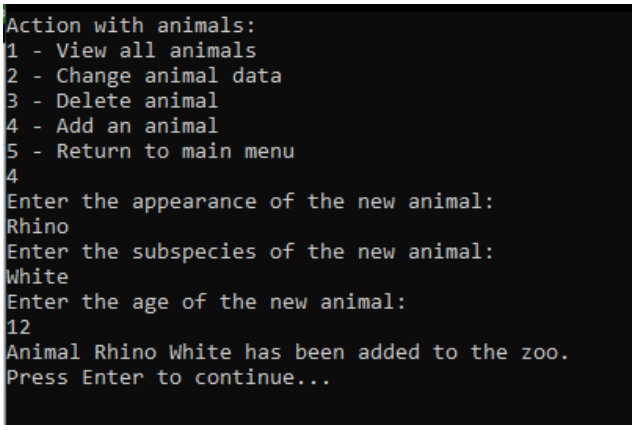
```
            Employees = new List<Employee>();
        }
    }
}
```



Мал.1. Головне меню

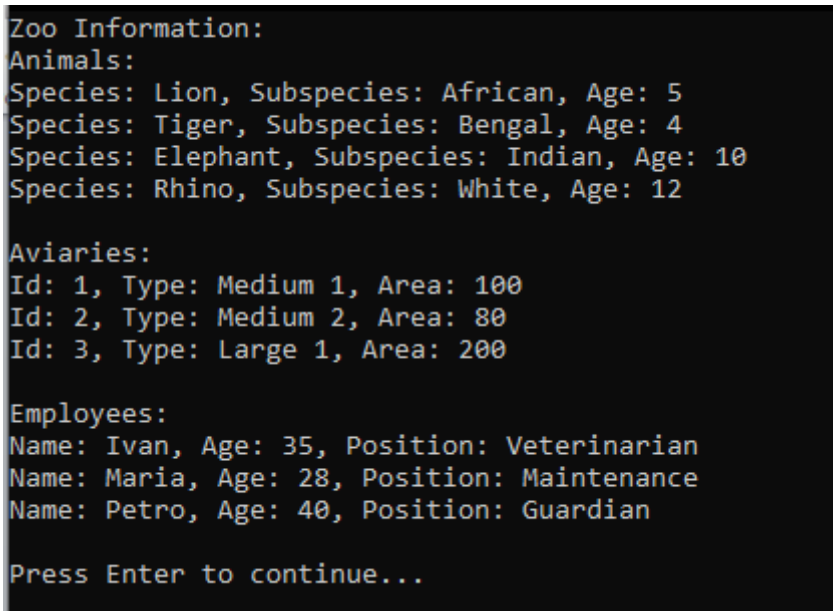


Мал.2. Перегляд тварин



Мал.3. Додавання нової тварини

За аналогією до тварин зроблені меню для вольєрів, працівників та продуктів.



Мал.4. Перегляд інвертизаціїї

Завдання № 3: Опишіть особливості дотримання принципів програмування в Вашому коді

1. Додайте файл README.md в кореневу директорію цієї лабораторної роботи. В файлі README.md опишіть дотримання окремо кожного принципу програмування, який Вам відомо, і який можна продемонструвати Вашим кодом.

2. Опис можна залишати українською або (бажано) англійською мовами.

3. Опис повинен містити посилання на відповідні файли і рядки коду.

4. Для отримання максимальної оцінки потрібно продемонструвати мінімум 7 принципів. SOLID принципи рахуються окремо. Повний список принципів, які було розглянуто на лекції:

a. DRY,

b. KISS,

c. SOLID (5 окремих принципів)

d. YAGNI

e. Composition Over Inheritance

f. Program to Interfaces not Implementations

g. Fail Fast

B. KISS (Keep It Simple, Stupid):
The KISS principle implies that the system should be as simple and understandable as possible. In this code, we try to keep the logic simple, using simple data structures and clear methods. For example, the ViewAll method simply prints information about the zoo to the console making the code clear and easy to understand.
You can marvel at the butt here [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L31).

Single Responsibility Principle: Each class is responsible for only one piece of functionality. For example, the Zoo, Animal, Enclosure, Employee, and Food classes are responsible for managing their respective zoo entities.
You can marvel at the butt here [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L114).

Open/Closed Principle: Classes are written to be open to extension but closed to modification. For example, you can easily add new methods or properties to classes without changing existing code.
You can marvel at the butt here [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L78).

D. YAGNI (You Aren't Gonna Need It): The YAGNI principle suggests not adding functionality that is not currently needed. There is no obvious violation of this principle in the code, since functionality is added only when necessary (for example, methods for adding, removing, and viewing elements).

E. Composition Over Inheritance: The principle is that it is better to use object composition than class inheritance. There are no examples of inheritance in the code above, but there are examples of object composition, for example, Zoo contains the lists Animals, Enclosures, Employees, and Foods.
You can marvel at the butt here [Classes.cs](KPZ-LAB-01/ZooClassLibrary/Classes.cs#L29).

F. Program to Interfaces not Implementations: The code uses an interface-based approach, which allows for more flexible dependency management and easier testing. For example, methods for working with Zoo objects accept the Zoo interface instead of a specific implementation, making it easy to replace implementations as needed.
You can marvel at the butt here [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L78).

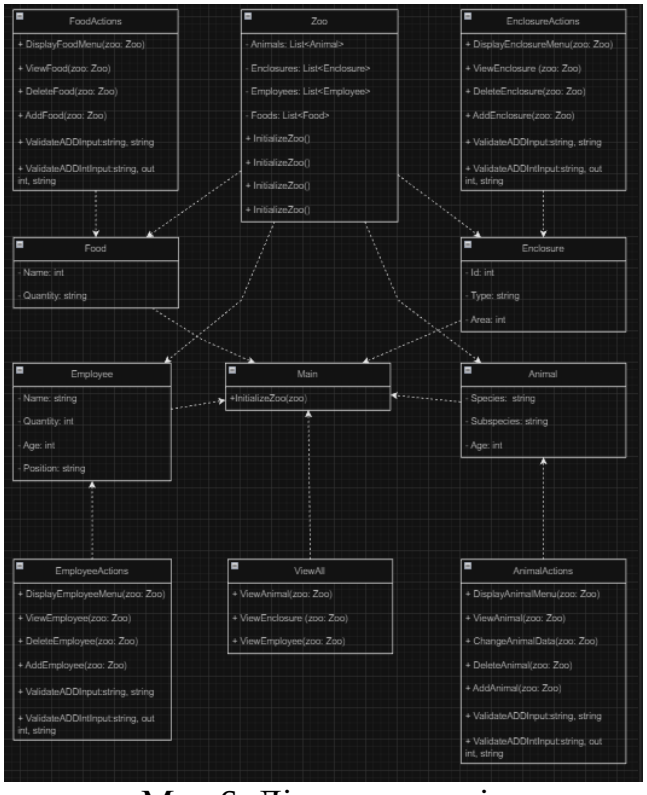
G. Fail Fast: The principle is to detect and respond to errors as early as possible. The code contains checks for user input and checks for null references, which is in line with the Fail Fast principle.
For this purpose the methods ValidateADDInput [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L524) and ValidateADDIntInput [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L533) were created, and the results of the processing are for example in the fragment [Program.cs](KPZ-LAB-01/KPZ-LAB-01/Program.cs#L490).

Мал.5. Створений файл README

Завдання №4: UML діаграма

1. Підготувати діаграму створених у програмі класів та інтерфейсів за допомогою https://app.diagrams.net/.

2. Експортувати створену діаграму та запушити експортований файл у кореневу директорію цією лабораторної



Мал.6. Діаграма класів

Висновки: я навчився дотримуватися принципів програмування та обґрунтовувати їх.