

Лабораторна робота №1

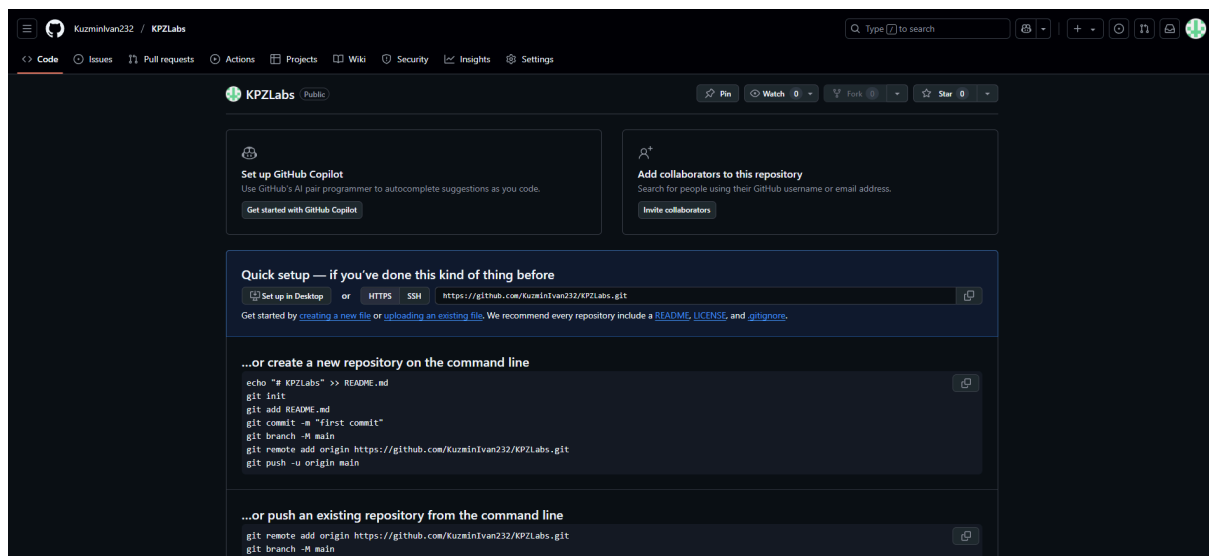
Тема: Принципи програмування. DRY, KISS, SOLID, YAGNI та ін.

Мета роботи: навчитися дотримуватися принципів програмування та обґрунтовувати їх.

Завдання 0: Підготовка до виконання завдання

1. Створити окремий **публічний** репозиторій на GitHub. В цьому репозиторії будуть міститися всі виконані Вами завдання курсу “Конструювання програмного забезпечення”. Кожне завдання буде міститися в окремій директорії або гілці **lab-1**, **lab-2 ... lab-n**
2. Впевнитися, що Ви уважно прочитали пункт 1 і Ваш репозиторій точно **публічний** ;)
3. Склонувати створений репозиторій
4. Обрати **один з трьох** варіантів Завдання 1 😊

Результат виконання:



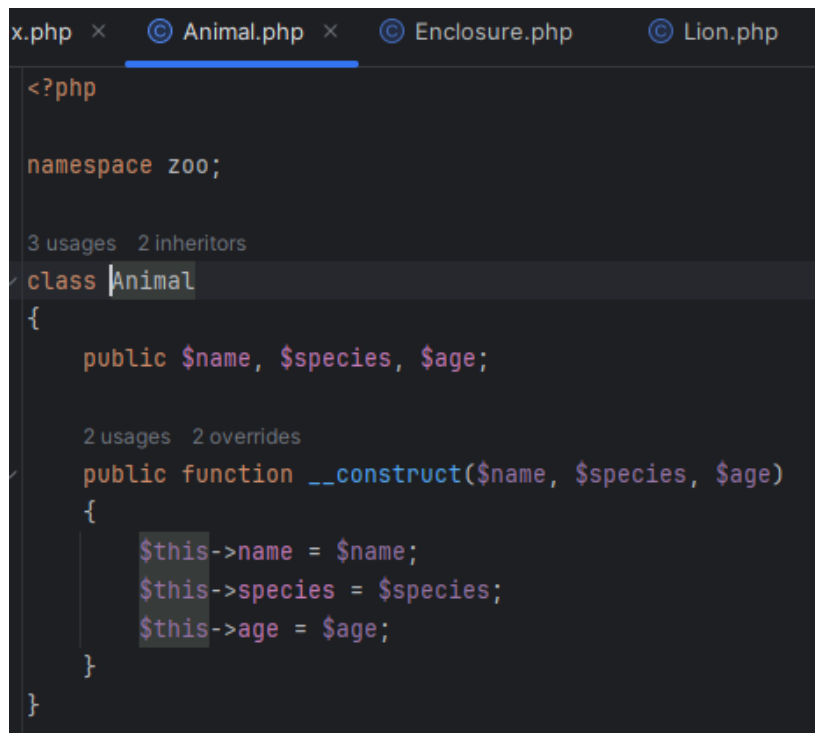
Завдання 1 Виконати завдання з дотриманням відомих Вам принципів програмування. **Один** на вибір з 3 варіантів.

Варіант 2: Зоопарк

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.

Результат виконання:

Клас Animal:



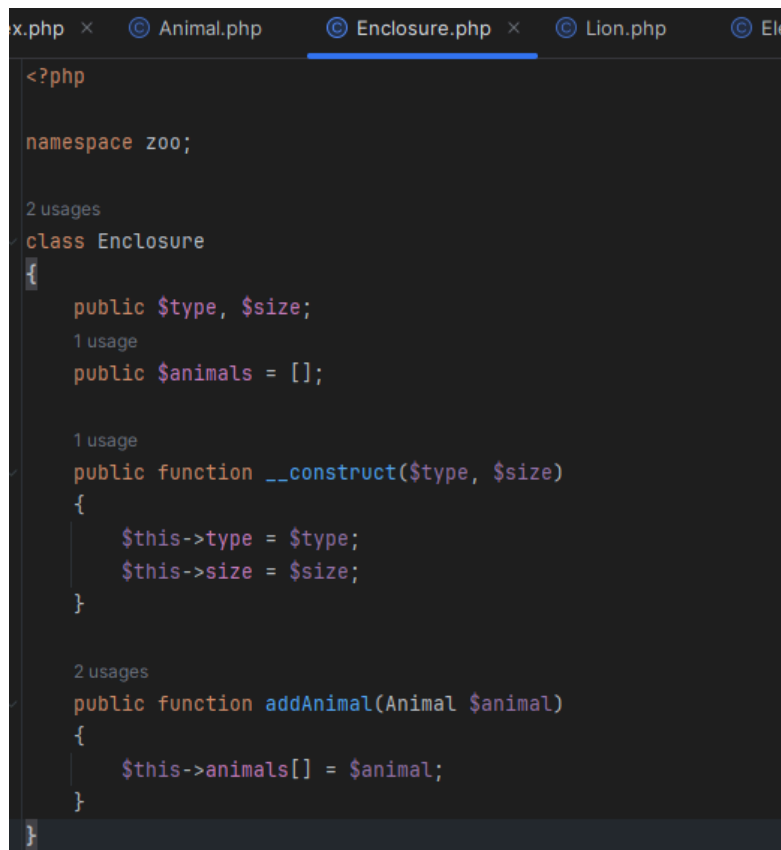
```
<?php

namespace zoo;

3 usages 2 inheritors
class Animal
{
    public $name, $species, $age;

    2 usages 2 overrides
    public function __construct($name, $species, $age)
    {
        $this->name = $name;
        $this->species = $species;
        $this->age = $age;
    }
}
```

Клас Enclosure:



```
<?php

namespace zoo;

2 usages
class Enclosure
{
    public $type, $size;
    1 usage
    public $animals = [];

    1 usage
    public function __construct($type, $size)
    {
        $this->type = $type;
        $this->size = $size;
    }

    2 usages
    public function addAnimal(Animal $animal)
    {
        $this->animals[] = $animal;
    }
}
```

Клас Food:

```
ex.php x Animal.php Enclosure.php Food.php x
<?php
namespace zoo;

3 usages
class Food
{
    public $name, $quantity; // quantity in kg

    2 usages
    public function __construct($name, $quantity)
    {
        $this->name = $name;
        $this->quantity = $quantity;
    }
}
```

Клас Employee:

```
<?php
namespace zoo;

3 usages
class Employee
{
    public $name, $position;

    2 usages
    public function __construct($name, $position)
    {
        $this->name = $name;
        $this->position = $position;
    }
}
```

Клас Inventory:

```
<?php

namespace zoo;

class Inventory
{
    public $enclosures = [];
    public $employees = [];
    public $food = [];

    public function addEnclosure(Enclosure $enclosure)
    {
        $this->enclosures[] = $enclosure;
    }

    public function addEmployee(Employee $employee)
    {
        $this->employees[] = $employee;
    }

    public function addFood(Food $food)
    {
        $this->food[] = $food;
    }

    public function report()
    {
        echo "Animals: <br>";
        foreach ($this->enclosures as $enclosure) {
            foreach ($enclosure->animals as $animal) {
                echo "{$animal->species} - {$animal->name},
{$animal->age} years <br>";
            }
        }

        echo "Employee: <br>";
        foreach ($this->employees as $e) {
            echo "{$e->name} - {$e->position} <br>";
        }

        echo "Food: <br>";
        foreach ($this->food as $f) {
            echo "{$f->name}: {$f->quantity} kg <br>";
        }
    }
}
```

Клас Lion:

```
<?php

namespace animals;

1 usage
class Lion extends \zoo\Animal
{
    1 usage
    public function __construct($name, $age)
    {
        parent::__construct($name, species: "Lion", $age);
    }
}
```

Клас Elephant:

```
<?php

namespace animals;

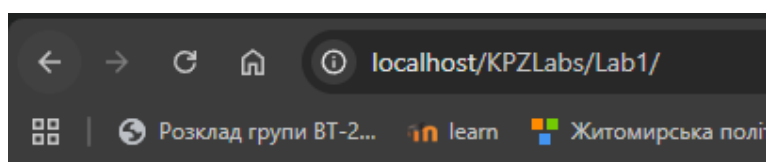
1 usage
class Elephant extends \zoo\Animal
{
    1 usage
    public function __construct($name, $age)
    {
        parent::__construct($name, species: "Elephant", $age);
    }
}
```

Завдання 2: Написати код для тестування отриманої функціональності.

1. Покажіть правильність роботи свого коду запустивши його в головному методі програми.
2. Достатньо буде просто вивести певну інформацію, щоб показати, що класи комунікують певним чином між собою.

Результат виконання:

```
php index.php x Animal.php Enclosure.php Lion.php Elephant.php
1 <?php
2
3 require_once "zoo/Inventory.php";
4 require_once "zoo/Enclosure.php";
5 require_once "zoo/Food.php";
6 require_once "zoo/Employee.php";
7 require_once "zoo/Animal.php";
8 require_once "animals/Lion.php";
9 require_once "animals/Elephant.php";
10
11 $zoo = new \zoo\Inventory();
12
13 $lion1 = new \animals\Lion( name: "Simba", age: 5);
14 $elephant1 = new \animals\Elephant( name: "Dambo", age: 10);
15
16 $enclosure1 = new zoo\Enclosure( type: "Savannah", size: "Large");
17 $enclosure1->addAnimal($lion1);
18 $enclosure1->addAnimal($elephant1);
19 $zoo->addEnclosure($enclosure1);
20
21 $zoo->addEmployee(new zoo\Employee( name: "Ivan", position: "Medic"));
22 $zoo->addEmployee(new zoo\Employee( name: "Olena", position: "Spectator"));
23
24 $zoo->addFood(new zoo\Food( name: "Meat", quantity: 50));
25 $zoo->addFood(new zoo\Food( name: "Fruits", quantity: 30));
26
27 $zoo->report();
```



Animals:

Lion - Simba, 5 years

Elephant - Dambo, 10 years

Employee:

Ivan - Medic

Olena - Spectator

Food:

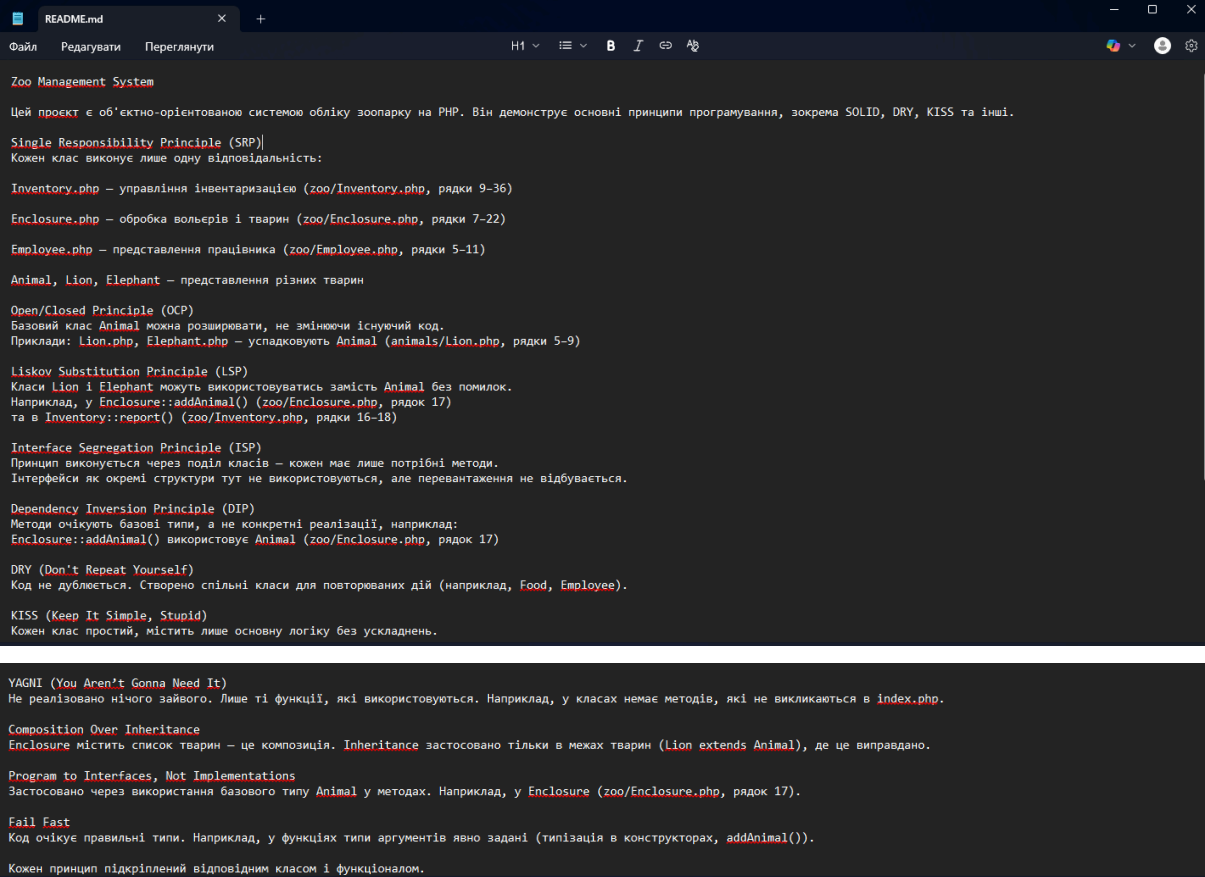
Meat: 50 kg

Fruits: 30 kg

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. Ваш опис повинен містити посилання на відповідні файли і рядки коду.
4. Як залишати посилання на свої рядки коду можна глянути [тутечки](#) (для посилання на директорію) або [тут](#) (для посилання на окремі рядки).
5. Синтаксис .md файлів документації можна знайти [туть](#) або [туть](#).
6. Для отримання максимальної оцінки Ви повинні 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

Результат виконання:



```
README.md

Zoo Management System

Цей проект є об'єктно-орієнтованою системою обліку зоопарку на PHP. Він демонструє основні принципи програмування, зокрема SOLID, DRY, KISS та інші.

Single Responsibility Principle (SRP)
Кожен клас виконує лише одну відповідальність:

Inventory.php – управління інвентаризацією (zoo/Inventory.php, рядки 9-36)
Enclosure.php – обробка вольєрів і тварин (zoo/Enclosure.php, рядки 7-22)
Employee.php – представлення працівника (zoo/Employee.php, рядки 5-11)
Animal, Lion, Elephant – представлення різних тварин

Open/Closed Principle (OCP)
Базовий клас Animal можна розширювати, не змінюючи існуючий код.
Приклади: Lion.php, Elephant.php – успадковують Animal (animals/Lion.php, рядки 5-9)

Liskov Substitution Principle (LSP)
Класи Lion і Elephant можуть використовуватись замість Animal без помилок.
Наприклад, у Enclosure::addAnimal() (zoo/Enclosure.php, рядок 17)
та в Inventory::report() (zoo/Inventory.php, рядки 16-18)

Interface Segregation Principle (ISP)
Принцип виконується через поділ класів – кожен має лише потрібні методи.
Інтерфейси як окремі структури тут не використовуються, але переваження не відбувається.

Dependency Inversion Principle (DIP)
Методи очікують базові типи, а не конкретні реалізації, наприклад:
Enclosure::addAnimal() використовує Animal (zoo/Enclosure.php, рядок 17)

DRY (Don't Repeat Yourself)
Код не дублюється. Створено спільні класи для повторюваних дій (наприклад, Food, Employee).

KISS (Keep It Simple, Stupid)
Кожен клас простий, містить лише основну логіку без ускладнень.

YAGNI (You Aren't Gonna Need It)
Не реалізовано нічого зайвого. Лише ті функції, які використовуються. Наприклад, у класах немає методів, які не викликаються в index.php.

Composition Over Inheritance
Enclosure містить список тварин – це композиція. Inheritance застосовано тільки в межах тварин (Lion extends Animal), де це виправдано.

Program to Interfaces, Not Implementations
Застосовано через використання базового типу Animal у методах. Наприклад, у Enclosure (zoo/Enclosure.php, рядок 17).

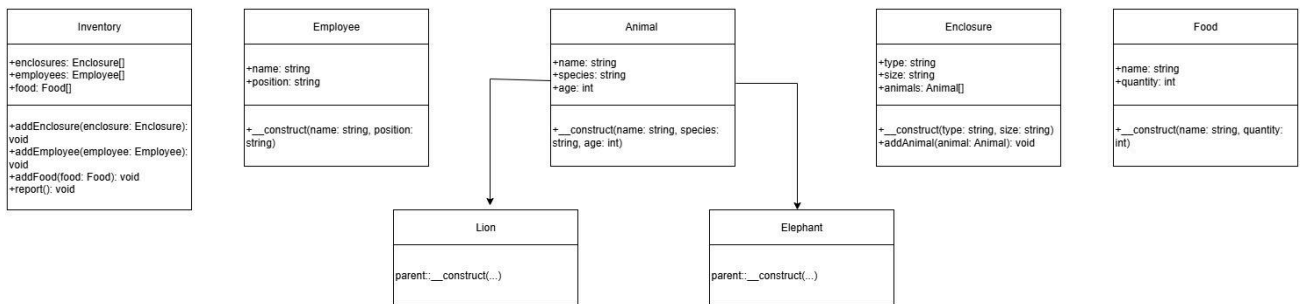
Fail Fast
Код очікує правильні типи. Наприклад, у функціях типи аргументів явно задані (типізація в конструкторах, addAnimal()).

Кожен принцип підкріплений відповідним класом і функціоналом.
```


Завдання 4: UML діаграма

1. Підготувати діаграму створених у програмі класів та інтерфейсів за допомогою <https://app.diagrams.net/>. **Звертайте особливу увагу на стрілки між сутностями.** На лекції було розглянуто особливості [створення UML діаграм](#) для нашого предмету.
2. Експортувати створену діаграму у вигляді картинки або PDF та запусити експортований файл у кореневу директорію цієї лабораторної.

Результат виконання:



Посилання на репозиторій: <https://github.com/KuzminIvan232/KPZLabs>