Zoo Homework

Homework Assignment: Creating a Zoo Management System

Instructions:

You are tasked with building a simple Zoo Management System using Object-Oriented Programming in Python. The system should be able to manage different types of animals in the zoo. You need to create classes for the following entities:

1. `Animal`: The base class for all animals. It should have the following attributes:
   - `name` (str): The name of the animal.
   - `species` (str): The species of the animal.
2. `Mammal`: A subclass of `Animal` for mammals. It should have an additional attribute:
   - `fur_color` (str): The color of the mammal's fur.
3. `Bird`: A subclass of `Animal` for birds. It should have an additional attribute:
   - `wing_span` (float): The wingspan of the bird.
4. `Reptile`: A subclass of `Animal` for reptiles. It should have an additional attribute:
   - `scale_type` (str): The type of scales the reptile has.
5. `Zoo`: A class to manage the animals in the zoo. It should have the following methods:
   - `add_animal(animal)`: Add an animal to the zoo.
   - `list_animals()`: List all the animals in the zoo.
   - `get_animals_by_species(species)`: List animals of a specific species in the zoo.
   - `remove_animal(name)`: Remove an animal from the zoo by its name.
   - `count_animals()`: Return the total number of animals in the zoo.

Requirements:

1. Create the necessary classes (`Animal`, `Mammal`, `Bird`, `Reptile`, and `Zoo`) with their attributes and methods.

2. Implement proper encapsulation by making attributes private and providing appropriate getters and setters.
3. Write a `main` program that demonstrates the functionality of the Zoo Management System. Create some animals of different types and perform operations such as adding, listing, and removing animals from the zoo.
4. Ensure that your code is well-documented, and you include comments explaining the purpose of classes, methods, and any important code blocks.

Bonus Challenge (Optional):

Create a method `feed_animals()` in the `Zoo` class that simulates feeding all the animals in the zoo. Each type of animal may have a different feeding mechanism, and you can implement this method to showcase polymorphism.

Grading:

- Correct implementation of classes and methods.
- Proper encapsulation and use of getter and setter methods.
- Demonstrated functionality in the main program.
- Code quality and documentation.

Домашнее задание: Создание системы управления зоопарком

Инструкция:

Вам предстоит создать простую систему управления зоопарком с использованием объектно-ориентированного программирования в Python. Система должна быть способна управлять разными типами животных в зоопарке. Вам нужно создать классы для следующих сущностей:

1. `Animal`: Базовый класс для всех животных. У него должны быть следующие атрибуты:
   - `name` (str): Имя животного.
   - `species` (str): Вид животного.

2. `Mammal`: Подкласс `Animal` для млекопитающих. У него должен быть дополнительный атрибут:
    - `fur_color` (str): Цвет шерсти млекопитающего.
3. `Bird`: Подкласс `Animal` для птиц. У него должен быть дополнительный атрибут:
    - `wing_span` (float): Размах крыльев птицы.
4. `Reptile`: Подкласс `Animal` для рептилий. У него должен быть дополнительный атрибут:
    - `scale_type` (str): Тип чешуи у рептилии.
5. `Zoo`: Класс для управления животными в зоопарке. У него должны быть следующие методы:
    - `add_animal(animal)`: Добавить животное в зоопарк.
    - `list_animals()`: Показать список всех животных в зоопарке.
    - `get_animals_by_species(species)`: Показать животных определенного вида в зоопарке.
    - `remove_animal(name)`: Удалить животное из зоопарка по имени.
    - `count_animals()`: Вернуть общее количество животных в зоопарке.

Требования:

1. Создайте необходимые классы (`Animal`, `Mammal`, `Bird`, `Reptile` и `Zoo`) с их атрибутами и методами.
2. Реализуйте правильную инкапсуляцию, сделав атрибуты приватными и предоставив соответствующие геттеры и сеттеры.
3. Напишите программу `main`, которая демонстрирует функциональность системы управления зоопарком. Создайте несколько животных разных видов и выполняйте операции, такие как добавление, список и удаление животных из зоопарка.
4. Убедитесь, что ваш код хорошо документирован, и включите комментарии, объясняющие назначение классов, методов и важных блоков кода.

Бонусное задание (по желанию):

Создайте метод `feed_animals()` в классе `Zoo`, который имитирует кормление всех животных в зоопарке. Каждый тип животного может иметь различный механизм кормления, и вы можете реализовать этот метод, чтобы продемонстрировать полиморфизм.

Оценка:

- Правильная реализация классов и методов.
- Правильная инкапсуляция и использование методов для доступа к атрибутам.
- Демонстрация функциональности в программе `main`.
- Качество кода и документация.

https://indify.co/widgets/live/quotes/K0k9jRxzcfS6QkgckWOs

```
class Animal {
    #name;
    #species;

    constructor(name, species) {
        this.#name = name;
        this.#species = species;
    }

    get name() {
        return this.#name;
    }

    get species() {
```

```
        return this.#species;
    }

    print() {
        //console.log(`Name ${animal.name} species ${animal.spec
        return `${this.#name} ${this.#species}`;
    }
}

class Mammal extends Animal {
    #furColor;

    constructor(name, furColor) {
        super(name, "Mammal");
        this.#furColor = furColor;
    }

    get furColor() {
        return this.#furColor;
    }

    print() {
        let a = super.print();
        a += ` ${this.#furColor}`;
        console.log(a);
    }
}

class Bird extends Animal {
    #wingSpan;

    constructor(name, wingSpan) {
        super(name, "Bird");
        this.#wingSpan = wingSpan;
    }
```

```
    get wingSpan() {
        return this.#wingSpan;
    }

    print() {
        let a = super.print();
        a += ` ${this.#wingSpan}`;
        console.log(a);
    }
}

class Reptile extends Animal {
    #scaleType;

    constructor(name, scaleType) {
        super(name, "Reptile");
        this.#scaleType = scaleType;
    }

    get scaleType() {
        return this.#scaleType;
    }

    print() {
        let a = super.print();
        a += ` ${this.#scaleType}`;
        console.log(a);
    }
}

class Zoo {
    #animals;

    constructor() {
        this.#animals = [];
    }
```

```
    addAnimal(animal) {
        this.#animals.push(animal);
    }

    listAnimals() {
        this.#animals.forEach((a) => this.printAnimal(a));
    }

    printAnimal(animal) {
        console.log("\n" + animal.print());
    }

    getAnimalsBySpecies(species) {
        this.#animals
            .filter((a) => a.species === species)
            .forEach((a) => this.printAnimal(a));
    }

    removeAnimal(name) {
        this.#animals.splice(
            this.#animals.findIndex((a) => a.name === name),
            1
        );
    }

    countAnimals() {
        return this.#animals.length;
    }
}

const z1 = new Zoo();

const m1 = new Mammal("Squirrel", "brown");
z1.addAnimal(m1);
```

```
const m2 = new Mammal("Polar bear", "white");
z1.addAnimal(m2);

const m3 = new Mammal("Wolf", "dark grey");
z1.addAnimal(m3);

const m4 = new Mammal("Fox", "red");
z1.addAnimal(m4);

const b1 = new Bird("Sparrow", "brown");
z1.addAnimal(b1);

const b2 = new Bird("Pigeon", "blue grey");
z1.addAnimal(b2);

const b3 = new Bird("Crow", "black");
z1.addAnimal(b3);

const r1 = new Reptile("Lizard", "platelike");
z1.addAnimal(r1);

const r2 = new Reptile("Snake", "smooth");
z1.addAnimal(r2);

const r3 = new Reptile("Turtle", "scutes");
z1.addAnimal(r3);

z1.listAnimals();
console.log(z1.countAnimals());
z1.removeAnimal("Sparrow");
console.log(z1.countAnimals());
console.log("\n");
z1.getAnimalsBySpecies("Bird");
```