OOPS - Encapsulation & Inheritance

IIT Ropar - Minor in AI

7th Feb, 2025

1 Enchanted Animal Sanctuary

Welcome to the Enchanted Animal Sanctuary, a magical place where various creatures coexist harmoniously. The sanctuary's caretakers need a system to manage their unique inhabitants. Let's explore how Object-Oriented Programming (OOP) can help create an efficient and secure management system.

2 The Challenge

The sanctuary needs to:

1. Keep track of different animals
2. Protect sensitive information about the creatures
3. Allow for easy addition of new animal types

3 The Solution: Magical Creature Management System

Let's dive into our Python-based solution that showcases encapsulation and inheritance.

```python
class Animal:
    def __init__(self, name):
        self._name = name    # Protected member
        self.__species = "Magical Creature"  # Private member

    def get_species(self):   # Public method to access private member
        return self.__species

class Dog(Animal):
    def __init__(self, name):
        super().__init__(name)
        self.__species = "Enchanted Canine"  # Private member specific to Dog

    def get_species(self):   # Public method to access private member
        return self.__species

    def bark(self):
        return f"{self._name} says Woof with a magical echo!"

# Let's add a new inhabitant to our sanctuary
buddy = Dog("Buddy")

print(f"Inhabitant name: {buddy._name}")
print(buddy.bark())
print(f"Species: {buddy.get_species()}")
```

OUTPUT:
Inhabitant name: Buddy
Buddy says Woof with a magical echo!
Species: Enchanted Canine

4 Explaining the Magic

4.1 Encapsulation: Protecting Our Magical Secrets

In our `Animal` class:

- `self._name` is protected (single underscore). It's like a light invisibility cloak – accessible but signaling "handle with care".
- `self.__species` is private (double underscore). It's under a powerful concealment charm, hidden from direct outside access.

Why is this important? Imagine if anyone could change an animal's species at will – chaos in the sanctuary!

4.2 Inheritance: Passing Down Magical Traits

Our `Dog` class inherits from `Animal`:

- It gets all the basic "magical creature" properties.
- It adds its own twist with a special bark and a unique species.

This is like magical genetics – dogs inherit general animal traits but have their own special abilities.

4.3 Method Overriding: Customizing Our Magic

The Dog class overrides the get_species method: This is like a specialized enchantment overriding the general one from Animal. This override changes the behavior of the get_species method for Dog objects:

- In the Animal class, it returns Magical Creature.
- In the Dog class, it returns Enchanted Canine.

4.4 Accessing Our Magical Properties

```
buddy = Dog("Buddy")
print(buddy.bark())
# Output: Buddy says Woof with a magical echo!
```

Here, we're calling a public method. It's like asking Buddy to perform a trick – anyone can do this!

```
print(f"Species: {buddy.get_species()}")
# Output: Species: Enchanted Canine
```

We're using a public method to access a private attribute. It's like using a special spell to reveal hidden information.

```
print(f"Name: {buddy._name}")  # Output: Name: Buddy
```

Directly accessing a protected member. It works, but it's frowned upon – like using a forbidden spell.

```
# print(buddy.__species)  # This would raise an AttributeError
```

Trying to access a private member directly. It's like attempting to break through a powerful protection spell – it just won't work!

5 Flying Twist

5.1 Adding a new inhabitant

A pigeon named "pichku" flies into our sanctuary and insists upon staying here. We then add pigeons too into our system.

```
class Pigeon(Animal):
    def __init__(self, name):
        super().__init__(name)
        self.__species = "Neighbourhood Pigoen"
        self.flies = True

    def get_species(self):
        return self.__species

    def coo(self):
        return f"{self._name} says Coo with a magical echo!"

# Let's add a new inhabitant to our sanctuary
pichku = Pigeon("Pichku")

print(f"Inhabitant name: {pichku._name}")
print(pichku.coo())
print(f"Species: {pichku.get_species()}")
print(f"Can fly: {pichku.flies}")
```

OUTPUT:
Inhabitant name: Pichku
Pichku says Coo with a magical echo!
Species: Neighbourhood Pigoen
Can fly: True

5.2 Finding Flying Inhabitants

Now, we create a function to find total number of inhabitants that can fly in our sanctuary.

```
# function to count number of Animals that can fly
def count_flying_animals(animals):
    count = 0
    for animal in animals:
        if hasattr(animal, 'flies') and animal.flies:
            count += 1
    return count

```

```python
# Current inhabitants of our sanctuary
buddy = Dog("Buddy")
pichku = Pigeon("Pichku")
brownie = Dog("Brownie")
tweety = Pigeon("Tweety")

# list of all inhabitants
animalsList = [buddy, pichku, brownie, tweety]

# count of flying inhabitants
print(count_flying_animals(animalsList))
```

OUTPUT:
2

There are total 2 flying inhabitants in our sanctuary currently.

6 The Results

Our Magical Creature Management System successfully:

1. Keeps track of different animals (like Buddy, Pichku)
2. Protects sensitive information (the private `__species`)
3. Allows for easy addition of new animal types (we can create more classes inheriting from `Animal`) through code reusability.

7 Your Turn: Expand the Sanctuary

Now that you've seen how our Magical Creature Management System works, it's time for you to add your own enchantment to the sanctuary!

7.1 Challenge: Introduce Magical Felines

The sanctuary has decided to welcome a new type of inhabitant: Enchanted Cats. Your mission, should you choose to accept it, is to create a `Cat` class similar to our `Dog` class.

Here's what you need to do:

1. Create a new class named `Cat` that inherits from `Animal`.
2. Initialize it with a name, just like the `Dog` class.
3. Set a private species attribute specific to cats.
4. Add a method called `meow()` that returns a string with the cat's name and a meow sound.

Here's a template to get you started:

```python
class Cat(Animal):
    pass

# Test your class here
# whiskers = Cat("Whiskers")
# print(whiskers.meow())
# print(f"Species: {whiskers.get_species()}")
```

7.2 Hints

- Remember to use `super().__init__(name)` to properly initialize the parent class.
- Use double underscores (__) for the private species attribute.
- Get creative with your magical meow sound!

Once you've implemented the `Cat` class, try creating a few cat instances and test out their methods. How does accessing their attributes compare to what we saw with the `Dog` class?

Try doing this challenge yourself rather than using AI tools to by comparing with the Dog class so, that you can reinforce your understanding of inheritance and encapsulation in Python.

8 Conclusion

Through the power of OOP, we've created a flexible and secure system for our Enchanted Animal Sanctuary. Encapsulation keeps our magical creatures' secrets safe, while inheritance allows us to easily expand our sanctuary with new types of enchanted animals.