Name: Hamsa M Ahmed
Subject: OOP Practical
Department: Computer Network System

Lecture 4: Encapsulation and Access Modifiers

1. Introduction

Encapsulation is a core concept in OOP that binds data and methods together, and restricts access to some of the object's components. This helps protect the integrity of data by preventing outside interference and misuse.

2. Encapsulation in Python

Python does not have strict access modifiers like `private` or `public` in other languages, but it follows conventions:

- Single underscore prefix `_var` indicates **protected** (should not be accessed directly).
- Double underscore prefix `__var` triggers **name mangling** to make the attribute harder to access from outside (similar to private).

3. Implementing Encapsulation

```
class Person:
    def __init__(self, name, age):
        self.__name = name      # private attribute
        self.__age = age

    def get_name(self):
        return self.__name

    def set_name(self, new_name):
        self.__name = new_name

    def get_age(self):
        return self.__age

    def set_age(self, new_age):
        if new_age > 0:
            self.__age = new_age
        else:
            print("Invalid age")
```

Name: Hamsa M Ahmed
Subject: OOP Practical
Department: Computer Network System

4. Usage Example

```
p = Person("Alice", 30)
print(p.get_name())  # Alice
p.set_age(35)
print(p.get_age())   # 35
p.set_age(-5)        # Invalid age
```

5. Why Use Encapsulation?

- Prevents accidental modification of data.
- Controls how data is accessed or modified.
- Supports validation logic inside setter methods.

6. Properties: Pythonic Way

Python provides the `@property` decorator to simplify getters and setters:

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, value):
        self.__name = value

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, value):
        if value > 0:
            self.__age = value
        else:
            print("Invalid age")
```

Name: Hamsa M Ahmed
Subject: OOP Practical
Department: Computer Network System

7. Exercises

1. Implement a `BankAccount` class with private balance. Provide deposit and withdrawal methods with validation.
2. Use `@property` to manage the balance attribute.
3. Test your class by performing deposits and withdrawals and printing the balance.

8. Summary

Encapsulation helps protect object integrity by restricting direct access to data and allowing controlled access via methods. Python uses naming conventions and the `@property` decorator to support encapsulation.

9. References

1. Python Official Docs - Properties
2. Real Python - Encapsulation