DOMINIC MUHIARWE
M23B13/024
B20865

Object Oriented Programming

Assignment

Part 1: Theoretical Questions

Qn.1:

Object-Oriented Programming is centered around the concept of objects" which can represent real-world entities or concepts. Object-Oriented Programming focuses on organizing code in a way that mimics real-world interactions

Four Main Pillars of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

So together, these four pillars promote better software design by:

- Enhancing Modularity: Code can be broken down into smaller, manageable pieces, making it easier to understand and maintain.
- Improving Reusability: Through encapsulation, inheritance, and polymorphism, developers can reuse code across different parts of a program or even across different projects.
- Facilitating Maintenance: Changes can be made to one part of the system with minimal impact on others, reducing the risk of introducing bugs.

Qn.2:

So the constructor is a special method used to initialize a newly created object. The __init__ method serves as the constructor and is automatically called when an object is instantiated. It allows you to set initial values for the object's attributes. Eg.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Qn.3

Class Variables: They are defined within the class but outside of any methods.
Instance Variables: Unique to each instance of a class. They are defined within the __init__ method and are prefixed with self. Eg:

```python
class Dog:
    species = "Ugandan Sheperd"  # Class variable

    def __init__(self, name, age):
        self.name = name  # Instance variable
        self.age = age    # Instance variable

# Creating instances of Dog
dog1 = Dog("Dave", 3)
dog2 = Dog("Andrew", 5)

print(dog1.species) # this prints Ugandan Sheperd
print(dog2.species) # this prints Ugandan Sheperd

# Changing instance variables
dog1.age = 4
print(dog1.age)  # Output: 4
print(dog2.age)  # Output: 5
```

So in this example, species is a class variable shared among all Dog instances, while name and age are instance variables unique to each Dog.

Qn.4

Class Method: A method that receives the class as the first argument and can modify class state that applies across all instances. It's defined using the @classmethod decorator.

Static Method: A method that does not receive any reference to the class or instance as its first argument. It's defined using the @staticmethod decorator and is used for utility functions that don't modify class or instance state.

```python
class Dog:
    species = "Ugandan Sheperd Innit"

    @classmethod
    def get_species(cls):
        return cls.species

    @staticmethod
    def bark():
        return "Woof!"

# Using the class method
print(Dog.get_species())  # Output: Ugandan Sheperd Innit

# Using the static method
print(Dog.bark())  # Output: Woof!
```

In this example, get_species is a class method that can access the class variable species, while bark is a static method that simply returns a string and doesn't depend on class or instance data.

Qn.5

Tracking the Total Number of Books

Imagine you have a library system where each time a new book is added to the catalog, you want to keep track of how many books exist in total. This total should be the same across all instances of the Book class, rather than being specific to each book.

Using a class variable for counting total instances is appropriate because it provides a clear, efficient, and centralized way to manage shared state across multiple objects.