📘 Python Polymorphism and Abstraction – Detailed Notes

🕰️ 1. Polymorphism

- **Polymorphism** means "many forms".
- Ability of **objects of different classes** to be treated as objects of a **common superclass**.
- Allows **same interface** to represent different underlying forms (data types or classes).

Types of Polymorphism

1. **Compile-Time (Method Overloading)**: Same method name with different parameters (not natively in Python but can simulate with default arguments).
2. **Run-Time (Method Overriding)**: Subclass provides specific implementation of method in parent class.

🕰️ 2. Example: Method Overriding (Run-Time Polymorphism)

```
class Vehicle:
    def start(self):
        print("Vehicle is starting")

class Car(Vehicle):
    def start(self):
        print("Car is starting")

class Bike(Vehicle):
    def start(self):
        print("Bike is starting")

v1 = Vehicle()
v2 = Car()
v3 = Bike()

for vehicle in [v1, v2, v3]:
    vehicle.start()
```

Output:

```
Vehicle is starting
Car is starting
Bike is starting
```

Real-Time: Vehicle management system handling cars, bikes, and trucks with same `start` interface.

🕰️3. Example: Polymorphism with Functions

```python
# Same function can process different data types
def add(a, b):
    return a + b

print(add(10, 20))        # 30 (Integers)
print(add("Hello, ", "World"))  # Hello, World (Strings)
print(add([1,2], [3,4]))  # [1,2,3,4] (Lists)
```

Real-Time: Payment system where `add` can handle numbers, strings (descriptions), or lists (items).

🕰️4. Abstraction

- **Abstraction** hides **complex implementation details** and shows only the **necessary functionality**.
- Achieved using **abstract classes** and **abstract methods** (from `abc` module).

Syntax

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass
```

🕰️5. Example: Abstraction with Shapes

```python
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

rect = Rectangle(10, 5)
print("Area:", rect.area())          # 50
print("Perimeter:", rect.perimeter()) # 30
```

Real-Time: Geometry calculator showing only required methods (`area` and `perimeter`) without exposing calculations internally.

🕰️6. Example: Abstraction in Banking System

```python
class BankAccount(ABC):
    @abstractmethod
    def deposit(self, amount):
        pass

    @abstractmethod
    def withdraw(self, amount):
        pass

class SavingsAccount(BankAccount):
    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient balance")

acc = SavingsAccount(1000)
acc.deposit(500)
```

```
acc.withdraw(300)
print(acc.balance)  # 1200
```

Real-Time: Bank software using abstraction to enforce deposit and withdraw methods for different account types.

🕰️7. Summary

- **Polymorphism:** Single interface, multiple forms. Improves flexibility and code reusability.
- **Abstraction:** Hides implementation details, shows only essential features. Achieved using abstract classes/methods.
- **Real-Time Uses:** Vehicle management, payment systems, banking systems, geometric calculators.

📥Download Options

- Download as DOCX
- Download as TXT
- Download as PDF