Polimorfismo en POO

Alumno: Ticona Miramira Roberto Angel
Docente: Ing. Coyla Idme Leonel
Lenguajes de Programación II – FINESI
Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Introducción

En la Programación Orientada a Objetos (POO), el **polimorfismo** es la capacidad que tienen los objetos de tomar distintas formas. Permite que un mismo método o función se comporte de manera diferente según el tipo de objeto que lo invoque. En Python, esto se logra mediante **herencia**, **redefinición de métodos** y el **duck typing** (tipado implícito).

1 Sobrescritura de Métodos

Ejemplo 1: Clases Animales

```python
class Animal:
    def hacer_sonido(self):
        print("Sonido gen rico")

class Perro(Animal):
    def hacer_sonido(self):
        print("Guau")

class Gato(Animal):
    def hacer_sonido(self):
        print("Miau")

animales = [Perro(), Gato(), Animal()]
for animal in animales:
    animal.hacer_sonido()
```

Listing 1: Polimorfismo mediante herencia

Ejecución:

```
Guau
Miau
Sonido genérico
```

Ejemplo 2: Clases Vehiculos

```
class Vehiculo:
    def iniciar(self):
        print("El vehiculo esta listo para moverse")

class Coche(Vehiculo):
    def iniciar(self):
        print("El coche arranca y acelera")

class Bicicleta(Vehiculo):
    def iniciar(self):
        print("La bicicleta empieza a pedalear")

class Barco(Vehiculo):
    def iniciar(self):
        print("El barco enciende el motor y zarpa")

vehiculos = [Coche(), Bicicleta(), Barco()]
for v in vehiculos:
    v.iniciar()
```

Listing 2: Método `iniciar()` polimórfico

Ejecución:

```
El coche arranca y acelera
La bicicleta empieza a pedalear
El barco enciende el motor y zarpa
```

2 Polimorfismo por Duck Typing

El duck typing permite que cualquier objeto con un método del mismo nombre sea usado, sin necesidad de herencia explícita.

Ejemplo 1: Clase Voladores

```
class Pajaro:
    def volar(self):
        print("El pajaro vuela")

class Avion:
    def volar(self):
        print("El avi n vuela")

def hacer_volar(obj):
    obj.volar()

hacer_volar(Pajaro())
hacer_volar(Avion())
```

Listing 3: Duck typing con método `volar()`

Ejecución:

```
El pajaro vuela
El avión vuela
```

Ejemplo 2: Clase Movimiento4

```python
class Pajaro:
    def mover(self):
        print("El pajaro vuela")

class Pez:
    def mover(self):
        print("El pez nada")

class Persona:
    def mover(self):
        print("La persona camina")

def desplazar(objeto):
    objeto.mover()

objetos = [Pajaro(), Pez(), Persona()]
for objeto in objetos:
    desplazar(objeto)
```

Listing 4: Método `mover()` en distintas clases

Ejecución:

```
El pajaro vuela
El pez nada
La persona camina
```

3 Ventajas del Polimorfismo

- **Flexibilidad y reutilización:** Permite escribir código que funciona con distintos tipos de objetos sin modificar la lógica.
- **Reducción de acoplamiento:** El código depende de interfaces (métodos), no de tipos específicos.
- **Uso de clases base abstractas:** En Python, aunque no hay interfaces formales como en Java, se pueden usar clases base para forzar la implementación de ciertos métodos.