Polimorfismo en el Sistema de Biblioteca: Profundización

🔍 ¿Qué es el Polimorfismo en POO?

El **polimorfismo** (del griego "muchas formas") permite que objetos de **diferentes clases** respondan al **mismo mensaje** (método) de forma particular. En nuestro sistema de biblioteca, se manifiesta cuando:

- Objetos `Libro`, `LibroDigital` y `LibroAudio` pueden ser tratados como `Libro`.
- Cada uno implementa sus propias versiones de métodos como `__str__()` o `prestar()`.

🛠️ Cómo se Implementa el Polimorfismo en el Ejemplo

1. Polimorfismo en Métodos Comunes

Código Base:

python

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

    def describir(self):
        return f"Libro físico: {self.titulo}"

class LibroDigital(Libro):
    def describir(self):  # Sobrescribe el método
        return f"E-book: {self.titulo} (PDF)"

class LibroAudio(Libro):
    def describir(self):  # Sobrescribe el método
        return f"Audiobook: {self.titulo} (MP3)"
```

Uso Polimórfico:

python

```
libros = [Libro("Cien años de soledad", "García Márquez"),
          LibroDigital("Python Crash Course", "Eric Matthes"),
```

```
            LibroAudio("El Principito", "Saint-Exupéry")]

for libro in libros:
    print(libro.describir())  # ¡Cada tipo responde diferente!
```

Salida:

```
text

Libro físico: Cien años de soledad
E-book: Python Crash Course (PDF)
Audiobook: El Principito (MP3)
```

2. Polimorfismo con Métodos Especiales (`__str__`)

```
python

class Libro:
    def __str__(self):
        return f"{self.titulo} - {self.autor}"

class LibroDigital(Libro):
    def __str__(self):
        return f"[DIGITAL] {super().__str__()}"

# Uso:
print(Libro("Rayuela", "Cortázar"))  # "Rayuela - Cortázar"
print(LibroDigital("Clean Code", "Robert Martin"))  # "[DIGITAL] Clean Code - Robert Mart
in"
```

⚡ ¿Cuándo Aplicar Polimorfismo?

✅ **Cuando hay comportamientos similares pero no idénticos**:

- Todos los libros se pueden describir, pero de forma distinta.

✅ **Cuando trabajas con colecciones heterogéneas**:

- Una lista que mezcla `Libro`, `LibroDigital`, etc.

✅ **Para extender funcionalidad sin modificar código existente**:

- Añadir `LibroAudio` sin cambiar cómo `Biblioteca` procesa los libros.

📌 Ejemplo Avanzado: Polimorfismo en Acciones de Biblioteca

python

```
class Biblioteca:
    def __init__(self):
        self.catalogo = []

    def agregar_libro(self, libro):
        self.catalogo.append(libro)

    def mostrar_catalogo(self):
        for libro in self.catalogo:
            print(libro)  # Llama al __str__() apropiado

    def exportar_formato(self):
        return [libro.exportar() for libro in self.catalogo]  # Polimorfismo

class Libro:
    def exportar(self):
        return f"FISICO|{self.titulo}"

class LibroDigital:
    def exportar(self):
        return f"DIGITAL|{self.titulo}|PDF"
```

🔎 Tipos de Polimorfismo Usados

1. Polimorfismo de Sobrescritura (Overriding)

- Las clases hijas redefinen métodos como `describir()` o `exportar()`.

2. Polimorfismo de Inclusión (Subtipo)

- `Biblioteca` trata a todos los libros igual, aunque sean subtipos diferentes.

3. Polimorfismo Paramétrico (Generics)

- No aplica directamente en Python (es dinámico), pero similar a aceptar cualquier `Libro`.

🎯 Beneficios Clave

Sin Polimorfismo	Con Polimorfismo
`if type(libro) == LibroDigital:` `...` (frágil)	`libro.describir()` (automático)
Código duplicado para cada tipo	Comportamiento unificado
Difícil añadir nuevos tipos	Fácil extensión

🚀 Ejemplo Real: Procesamiento de Préstamos

python

```
class Libro:
    def prestar(self):
        return "Préstamo físico registrado"

class LibroDigital:
    def prestar(self):
        return "Licencia temporal generada"

# Uso polimórfico:
def procesar_prestamo(libro):
    print(libro.prestar())  # ¡No importa el tipo!

procesar_prestamo(Libro("El Hobbit"))  # "Préstamo físico registrado"
procesar_prestamo(LibroDigital("Dune"))  # "Licencia temporal generada"
```

📚 Conclusión: Polimorfismo en la Biblioteca

- **Flexibilidad**: La biblioteca maneja nuevos tipos de libros sin cambios.
- **Limpieza**: Elimina condicionales (`if/elif`) para cada tipo.
- **Coherencia**: Todos los libros responden a la misma interfaz (`describir()` , `prestar()`).

Regla de Oro

> *"Programa hacia interfaces (métodos comunes), no hacia implementaciones concretas."*

El polimorfismo **hace tu código más adaptable** y **menos acoplado**📖➡️📱➡️🎧