Herencia en el Sistema de Biblioteca: Profundización

🔍 ¿Qué es la Herencia en POO?

La **herencia** es un mecanismo que permite crear nuevas clases (**clases hijas**) basadas en clases existentes (**clases padre**), reutilizando y extendiendo su funcionalidad. En nuestro sistema de biblioteca, `LibroDigital` hereda de `Libro`.

🛠️ Cómo se Implementa la Herencia en el Ejemplo de la Biblioteca

1. Relación entre `Libro` (Padre) y `LibroDigital` (Hija)

Cómo:

- `LibroDigital` hereda todos los atributos y métodos de `Libro` usando `super().__init__()`.
- **Extiende** la funcionalidad con nuevos atributos (`formato`, `tamanio_mb`) y métodos (`descargar()`).
- **Sobrescribe** el método `__str__` para personalizar la representación.

Código Base:

python

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

    def __str__(self):
        return f"{self.titulo} por {self.autor}"

class LibroDigital(Libro):
    def __init__(self, titulo, autor, paginas, isbn, formato, tamanio_mb):
        super().__init__(titulo, autor, paginas, isbn)  # Hereda atributos
        self.formato = formato  # Nuevo atributo
        self.tamanio_mb = tamanio_mb
```

```
    def descargar(self):  # Nuevo método
        return f"Descargando {self.titulo} en {self.formato}"

    def __str__(self):  # Sobrescritura
        return f"{super().__str__()} [Digital: {self.formato}]"
```

2. ¿Cuándo Usar Herencia?

✅ **Cuando existe una relación "es-un"**:

- `LibroDigital` **es un** `Libro` con características adicionales.

✅ **Cuando hay código duplicado**:

- Ambos tipos comparten `titulo`, `autor`, `paginas`, etc.

❌ **Cuando no hay relación lógica**:

- No heredarías `Biblioteca` de `Libro` (una biblioteca no es un libro).

3. ¿Por qué Usar Herencia Aquí?

1. **Reutilización**:
   - `LibroDigital` no repite código de `Libro` (ej: `titulo`, `__str__` base).
2. **Extensibilidad**:
   - Puedes añadir `LibroAudio` (hereda de `Libro`) sin modificar la clase padre.
3. **Polimorfismo**:
   - Tratar `Libro` y `LibroDigital` como iguales (ej: `biblioteca.agregar_libro()` acepta ambos).

⚡ Ejemplo: Polimorfismo en Acción

python

```
biblioteca = Biblioteca("Central")

libro_fisico = Libro("Cien años de soledad", "García Márquez", 432, "978123")
```

```
libro_digital = LibroDigital("Python Crash Course", "Eric Matthes", 544, "978456", "PDF",
15)

# Polimorfismo: Ambos son "Libros" para la biblioteca
biblioteca.agregar_libro(libro_fisico)
biblioteca.agregar_libro(libro_digital)

for libro in biblioteca.catalogo:
    print(libro)  # Llama a __str__() de cada tipo (físico o digital)
```

Salida:

```
text

Cien años de soledad por García Márquez
Python Crash Course por Eric Matthes [Digital: PDF]
```

🔎 Tipos de Herencia Usados

1. Herencia Simple

- `LibroDigital` hereda solo de `Libro` (una sola clase padre).

2. Sobrescritura de Métodos

- `LibroDigital` redefine `__str__` para añadir información del formato.

3. Extensión con `super()`

- `super().__init__()` llama al constructor de `Libro` antes de añadir atributos nuevos.

📌 Buenas Prácticas en la Herencia

✔ **Usar herencia solo cuando hay relación lógica** (no forzarla).
✔ **Evitar herencia múltiple** (puede ser confusa).
✔ **Documentar** qué métodos se sobrescriben o extienden.

🎯 ¿Qué Ganamos con Esta Herencia?

Sin Herencia	Con Herencia
`LibroDigital` repetiría `titulo`, `autor`, etc.	Reutiliza atributos/métodos de `Libro`.
No hay relación clara entre clases.	Jerarquía clara: `LibroDigital` es un `Libro`.
El polimorfismo no sería posible.	La biblioteca trata ambos tipos igual.

🚀 Ejemplo Avanzado: Herencia con Validación

python

```
class LibroAudio(Libro):
    def __init__(self, titulo, autor, paginas, isbn, duracion_horas, narrador):
        super().__init__(titulo, autor, paginas, isbn)
        self.duracion_horas = duracion_horas
        self.narrador = narrador

    def reproducir(self):
        return f"Reproduciendo {self.titulo} narrado por {self.narrador}"

    def __str__(self):
        return f"{super().__str__()} [Audio: {self.duracion_horas}h]"
```

Uso:

python

```
libro_audio = LibroAudio("El Principito", "Saint-Exupéry", 96, "978789", 2.5, "Pedro Pascal")
print(libro_audio.reproducir())  # Método exclusivo
biblioteca.agregar_libro(libro_audio)  # Polimorfismo
```

🔎 Conclusión: Herencia en la Biblioteca

- **Reutilización**: `LibroDigital` y `LibroAudio` aprovechan atributos/métodos de `Libro`.
- **Organización**: Jerarquía clara de tipos de libros.
- **Flexibilidad**: Puedes añadir más subtipos (`LibroInfantil`, `LibroCientifico`) sin modificar `Libro`.

Regla de Oro

> *"Usa herencia cuando la relación 'es-un' sea clara y cuando evites duplicar código".*

La herencia **estructura tu código** y lo hace **más mantenible**📚➡️📱