Programación Orientada a Objetos (POO) en Python: Explicación Detallada con Ejemplo Completo

La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el código en "objetos" que contienen datos y comportamientos relacionados. Python es un lenguaje que soporta completamente la POO.

Conceptos Fundamentales de POO

1. Clases y Objetos

- **Clase**: Es una plantilla o molde para crear objetos. Define los atributos y métodos que tendrán los objetos.
- **Objeto**: Es una instancia específica de una clase.

2. Pilares de la POO

- **Encapsulación**: Ocultar los detalles internos de cómo funciona un objeto.
- **Abstracción**: Representar solo las características esenciales.
- **Herencia**: Crear nuevas clases basadas en clases existentes.
- **Polimorfismo**: Capacidad de que objetos de diferentes clases respondan al mismo mensaje.

Ejemplo Completo: Sistema de Gestión de una Biblioteca

python

```
class Libro:
    # Atributo de clase (compartido por todas las instancias)
    contador_libros = 0

    def __init__(self, titulo, autor, paginas, isbn):
        # Atributos de instancia
        self.titulo = titulo
        self.autor = autor
        self.paginas = paginas
        self.isbn = isbn
        self.disponible = True
        Libro.contador_libros += 1
        self.id_libro = Libro.contador_libros

    # Método de instancia
    def prestar(self):
```

```
        if self.disponible:
            self.disponible = False
            return f"El libro '{self.titulo}' ha sido prestado."
        else:
            return f"El libro '{self.titulo}' no está disponible."

    def devolver(self):
        self.disponible = True
        return f"El libro '{self.titulo}' ha sido devuelto."

    # Método especial (dunder method)
    def __str__(self):
        return f"Libro {self.id_libro}: {self.titulo} por {self.autor}, {self.paginas} pá
ginas"

    # Método de clase
    @classmethod
    def get_total_libros(cls):
        return f"Total de libros en el sistema: {cls.contador_libros}"

    # Método estático
    @staticmethod
    def es_isbn_valido(isbn):
        return len(isbn) == 10 or len(isbn) == 13


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

    def agregar_libro(self, libro):
        if isinstance(libro, Libro):
            self.catalogo.append(libro)
            return f"Libro '{libro.titulo}' agregado a la biblioteca."
        else:
            return "Solo se pueden agregar objetos de tipo Libro."

    def buscar_libro(self, criterio, valor):
        resultados = []
        for libro in self.catalogo:
            if criterio == "titulo" and valor.lower() in libro.titulo.lower():
                resultados.append(libro)
            elif criterio == "autor" and valor.lower() in libro.autor.lower():
                resultados.append(libro)
            elif criterio == "isbn" and valor == libro.isbn:
                resultados.append(libro)
        return resultados

    def listar_libros(self):
        return [str(libro) for libro in self.catalogo]
```

```
# Herencia: LibroDigital hereda de Libro
class LibroDigital(Libro):
    def __init__(self, titulo, autor, paginas, isbn, formato, tamanio_mb):
        super().__init__(titulo, autor, paginas, isbn)
        self.formato = formato
        self.tamanio_mb = tamanio_mb
        self.descargas = 0

    def descargar(self):
        self.descargas += 1
        return f"Libro '{self.titulo}' descargado en formato {self.formato}. Tamaño: {sel
f.tamanio_mb}MB"

    # Overriding (sobrescritura de método)
    def __str__(self):
        return f"{super().__str__()} [Digital - {self.formato}, {self.tamanio_mb}MB]"


# Creación de objetos y uso del sistema
if __name__ == "__main__":
    # Crear una biblioteca
    biblio_central = Biblioteca("Biblioteca Central")

    # Crear algunos libros físicos
    libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", 432, "978030747472
8")
    libro2 = Libro("El Principito", "Antoine de Saint-Exupéry", 96, "9780156013925")

    # Crear un libro digital
    libro_digital1 = LibroDigital("Python Crash Course", "Eric Matthes", 544, "9781593279
288", "PDF", 15.5)

    # Agregar libros a la biblioteca
    print(biblio_central.agregar_libro(libro1))
    print(biblio_central.agregar_libro(libro2))
    print(biblio_central.agregar_libro(libro_digital1))

    # Prestar y devolver libros
    print(libro1.prestar())
    print(libro1.prestar())  # Intentar prestar nuevamente
    print(libro1.devolver())

    # Descargar libro digital
    print(libro_digital1.descargar())

    # Buscar libros
    print("\nBúsqueda por autor 'García':")
    resultados = biblio_central.buscar_libro("autor", "García")
    for libro in resultados:
        print(libro)
```

```
# Listar todos los libros
print("\nTodos los libros en la biblioteca:")
for libro_info in biblio_central.listar_libros():
    print(libro_info)

# Métodos de clase y estáticos
print("\n" + Libro.get_total_libros())
print("ISBN válido '1234567890':", Libro.es_isbn_valido("1234567890"))
print("ISBN válido '123':", Libro.es_isbn_valido("123"))
```

Explicación Detallada

1. Clase `Libro`

- **Atributos**:
  - `contador_libros` : Atributo de clase para llevar la cuenta de libros creados
  - `__init__` : Método constructor que inicializa los atributos de instancia
  - Atributos de instancia: `titulo` , `autor` , `paginas` , `isbn` , `disponible` , `id_libro`
- **Métodos**:
  - `prestar()` y `devolver()` : Métodos que modifican el estado del libro
  - `__str__()` : Método especial para representación en string
  - `get_total_libros()` : Método de clase que usa el atributo de clase
  - `es_isbn_valido()` : Método estático que no depende de la instancia

2. Clase `Biblioteca`

- Gestiona una colección de libros
- Métodos para agregar, buscar y listar libros
- Demuestra la relación entre clases (agregación)

3. Herencia con `LibroDigital`

- Hereda de `Libro` y añade nuevos atributos (`formato` , `tamanio_mb`)
- Sobrescribe el método `__str__` para incluir la nueva información
- Añade nuevos métodos específicos como `descargar()`

4. Conceptos Demostrados

- **Encapsulación**: Los detalles internos de las clases están ocultos

- **Abstracción**: Las clases representan conceptos del mundo real
- **Herencia**: `LibroDigital` extiende `Libro`
- **Polimorfismo**: Ambos tipos de libros responden a `__str__` pero de forma diferente

5. Métodos Especiales

- `__init__` : Constructor
- `__str__` : Representación en string
- `@classmethod` : Métodos de clase
- `@staticmethod` : Métodos estáticos