Practica 2: Programación Orientada a Objetos

Isidro Francisco Pérez Paz - 377806

Este documento describe la migración del sistema de biblioteca de C (estructurado) a Python (Orientado a Objetos), explicando los conceptos de POO aplicados, comparando ambas implementaciones y detallando las ventajas de la POO.

Conceptos de POO aplicados

¿Qué es una Clase?

Es una plantilla o molde para crear objetos. Define las propiedades (atributos) y los comportamientos (métodos) que los objetos de esa clase tendrán.

- **Ejemplo en el código:** La clase `Libro` es una plantilla. Define que cualquier libro tendrá un `_titulo`, `_autor`, `_anio_publicacion`, `_cantidad`, etc., y tendrá la capacidad (método) de `mostrar_detalles()`.

```
class Libro(ItemBiblioteca):
    def __init__(self, id_item, titulo, cantidad, autor, anio_publicacion,
genero):
        super().__init__(id_item, titulo, cantidad)
        self._autor = autor
        self._anio_publicacion = anio_publicacion
        self._genero = genero

    def mostrar_detalles(self):
        # ... implementación ...
```

¿Qué es un Objeto?

Es una instancia de una clase. Es una entidad concreta creada a partir de la plantilla (clase) que existe en la memoria y tiene sus propios valores para los atributos definidos por la clase.

- **Ejemplo en el código:** Cuando el administrador registra un libro, se crea un objeto de la clase `Libro` con datos específicos. `libro1` y `libro2` son objetos diferentes (instancias) de la misma clase `Libro`.

```
# En la función biblioteca.registrar_libro()
libro = Libro("101", "Cien Años de Soledad", 10, "G. García Márquez", 1967,
"Realismo Mágico")
# 'libro' es un objeto.
```

¿Qué es el Encapsulamiento?

Es el principio de ocultar el estado interno y la lógica de un objeto al mundo exterior, exponiendo solo una interfaz pública (métodos) para interactuar con él. Protege los datos de modificaciones indebidas.

- **Ejemplo en el código:** La clase `Biblioteca` encapsula las listas `_items` y `_usuarios`. El código del menú principal no accede directamente a `biblioteca._items.append(...)`. En su lugar, llama al método público `biblioteca.registrar_libro()`. Esto permite a la clase `Biblioteca` realizar validaciones (como chequear IDs duplicados) antes de añadir el ítem. El uso del guion bajo (`_items`) es una convención en Python para indicar que un atributo es "privado" o interno.

¿Qué es la Abstracción?

Consiste en mostrar solo las características esenciales de un objeto, ocultando los detalles complejos de su implementación. La clase ItemBiblioteca es una abstracción.

- **Ejemplo en el código:** Se definió una "Clase Base Abstracta" (ABC) llamada `ItemBiblioteca`. Esta clase decreta que cualquier ítem en la biblioteca (sea un libro, revista, o futuro DVD) debe tener un método `mostrar_detalles()`. Al código principal (como el método `mostrar_items` de la biblioteca) no le importa cómo un libro o una revista muestran sus detalles, solo sabe que puede llamar a `item.mostrar_detalles()`.

```
class ItemBiblioteca(ABC):
    # ...
    @abstractmethod
    def mostrar_detalles(self):
        pass # Fuerza a las subclases a implementar esto
```

¿Qué es la Herencia?

Es un mecanismo que permite a una clase (subclase o clase hija) heredar atributos y métodos de otra clase (superclase o clase padre). Esto promueve la reutilización de código.

- **Ejemplo en el código:** Tanto `Libro` como `Revista` son tipos de `ItemBiblioteca`. En lugar de repetir el código para `_id_item`, `_titulo`, `_cantidad`, `prestar()` y `devolver()` en ambas clases, estas propiedades y métodos se definen una vez en `ItemBiblioteca` y ambas clases hijas los heredan.

```
class Libro(ItemBiblioteca): # Libro HEREDA DE ItemBiblioteca
    # ...
class Revista(ItemBiblioteca): # Revista HEREDA DE ItemBiblioteca
    # ...
```

¿Qué es el Polimorfismo?

Literalmente "muchas formas". Es la capacidad de los objetos de diferentes clases de responder al mismo mensaje (llamada de método) de maneras diferentes.

- **Ejemplo en el código:** La herencia nos da el polimorfismo. Tenemos una lista de `ItemBiblioteca` que contiene objetos `Libro` y objetos `Revista`. Cuando iteramos sobre la lista y llamamos `item.mostrar_detalles()`:
  - Si `item` es un `Libro`, se ejecuta el `mostrar_detalles()` de `Libro` (mostrando autor y género).
  - Si `item` es una `Revista`, se ejecuta el `mostrar_detalles()` de `Revista` (mostrando editorial y número).
    El método `biblioteca.mostrar_items()` usa polimorfismo sin siquiera saber el tipo específico de cada ítem.

Conclusión

Usar la Programación Orientada a Objetos (POO) en este proyecto de biblioteca fue una gran ventaja porque hizo que el código fuera mucho más fácil de mantener, actualizar y entender que la versión anterior.
Gracias a la POO, el código está mejor organizado porque modela el mundo real (hay objetos para Libros, Usuarios y la Biblioteca misma). Esto hace que sea mucho más fácil añadir nuevos tipos de elementos, como revistas o DVDs, sin tener que duplicar código, lo que se conoce como escalabilidad. Además, esta organización reduce drásticamente la cantidad de errores y fallos en el programa. Finalmente, el proceso de guardar y cargar los datos es más seguro y robusto.