SOLID

Author: Popov Nichita

Group: FAF-222

Objectives:

- Get familiar with the SOLID principles;
- Implement 2 letters from SOLID in sample project;

Used Design Patterns:

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)

Implementation

In order to complete this laboratory work I implemented 2 letters from `SOLID`: Single Responsibility Principle (SRP) and Open/Closed Principle (OCP). The SRP is demonstrated by separating the responsibilities of storing book details and handling persistence (saving to a file) into different classes (`Book` and `BookPersistence`). The OCP is demonstrated by extending the `Book` class's functionality through a new `BookFormatter` class, which adds formatting methods without altering the existing `Book` implementation.

Single Responsibility Principle (SRP)

The **Single Responsibility Principle** states that a class should have only one reason to change, meaning it should have a single, focused responsibility. In the implementation, the `Book` class is responsible only for storing book details like title, author, and content.

```python
class Book:
    def __init__(self, title: str, author: str, content: str):
        self.title = title
        self.author = author
        self.content = content

    def get_summary(self) -> str:
        return f"'{self.title}' by {self.author}"
```

The `Book` class above handles only the data of the book, ensuring it adheres to SRP. To further uphold this principle, a separate class `BookPersistence` is used to handle the saving of the book's data to a file.

```python
class BookPersistence:
    @staticmethod
    def save_to_file(book: Book, filename: str) -> None:
        with open(filename, 'w') as file:
            file.write(f"Title: {book.title}\n")
            file.write(f"Author: {book.author}\n\n")
            file.write(book.content)
```

The `BookPersistence` class above has a single responsibility: saving the book data to a file. By separating it from the `Book` class, we ensure that each class has one reason to change, which makes the code more maintainable.

Open/Closed Principle (OCP)

The **Open/Closed Principle** states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This means we should be able to add new functionality without changing the existing code.

In the implementation, we add new functionality to the `Book` class by creating a separate `BookFormatter` class, which formats the book's content into different formats such as HTML or JSON-like representation.

```python
class BookFormatter:
    @staticmethod
    def format_as_html(book: Book) -> str:
        return f"<html><head><title>{book.title}</title></head><body><h1>{book.title}</h1><h

    @staticmethod
    def format_as_json(book: Book) -> dict:
        return {
            "title": book.title,
            "author": book.author,
            "content": book.content
        }
```

The `BookFormatter` class above provides new formatting capabilities for the `Book` class without modifying the original `Book` implementation. This adheres to the Open/Closed Principle by extending functionality in a separate class.

Example Usage

In the `main()` function, we can see how these classes work together to fulfill different responsibilities:

```python
def main():
    my_book = Book("The Python Handbook", "John Doe", "This is a comprehensive guide to Pyth
    BookPersistence.save_to_file(my_book, "my_book.txt")
    print(BookFormatter.format_as_html(my_book))
```

```
    print(BookFormatter.format_as_json(my_book))

if __name__ == "__main__":
    main()
```

The `main()` function creates an instance of `Book` and uses the `BookPersistence` and `BookFormatter` classes to save and format the book, demonstrating how SRP and OCP have been successfully applied.

Conclusions / Results

The implementation of the `Book`, `BookPersistence`, and `BookFormatter` classes demonstrates the practical application of two SOLID principles:

- **Single Responsibility Principle (SRP)**: Each class in the implementation has a single responsibility, reducing complexity and improving maintainability.
- **Open/Closed Principle (OCP)**: New functionality was added to extend the `Book` class (in the form of new formatting methods) without modifying the existing class, adhering to the OCP.

The use of these principles makes the code easier to understand, modify, and extend, ultimately leading to more robust and scalable software.