Ministry of Education, Culture, and Research of the Republic of Moldova
Technical University of Moldova
The Faculty of Computers, Informatics, and Microelectronics

REPORT

SOLID Principles

Laboratory work no. 0

Executed by: Cornievschi B.
st. gr. FAF-222

Verified by: Furdui A.
univ. assist.

Chişinău – 2024

1. Introduction

The purpose of this laboratory work is to demonstrate the implementation of two SOLID principles - the Single Responsibility Principle (SRP) and the Open/Closed Principle (OCP) - using a simple Java-based library management system. These principles aim to improve the maintainability, reusability, and scalability of code.

2. Overview of SOLID Principles

Single Responsibility Principle (SRP)

The Single Responsibility Principle states that a class should have only one reason to change, meaning it should only focus on a single task or responsibility. This helps in minimizing the potential for unexpected side-effects when changes are made. In our example, both the Book and Library classes follow SRP.

- **Book Class**: The Book class is responsible only for representing the details of a book, including title, author, and ISBN.
  - **Example**: The Book class has fields such as title, author, and isbn, along with getters to access these properties. This class does not deal with managing collections or filtering books.

- **Library Class**: The Library class is responsible for managing a collection of books, such as adding or removing books.
  - **Example**: The Library class has methods addBook() and removeBook(), which directly modify the list of books. It does not hold any book-specific information beyond the collection.

Open/Closed Principle (OCP)

The Open/Closed Principle states that software entities (such as classes or modules) should be open for extension but closed for modification. This means that new functionality should be added through extensions rather than altering existing code, thus maintaining stability while adding features.

In our example, we follow OCP by using the `BookFilter` interface and extending it to create different types of filters:

- **BookFilter Interface**: This interface defines a method filter(Book book) that can be implemented to provide custom filtering logic.
  - **Example**: The BookFilter interface serves as a base, allowing various filters to be implemented without modifying the existing code.

- **AuthorFilter Class**: Implements BookFilter to filter books by author name.
  - Example: The `AuthorFilter` class implements the filter() method to compare the author's name.

- **TitleFilter Class**: Implements BookFilter to filter books by title.
  - **Example**: The `TitleFilter` class implements the filter() method to compare the book title.

3. Implementation Details

The Java library system is implemented with a modular approach where each class has a specific responsibility:

- **Book Class**: Holds the book details such as `title`, `author`, and `isbn`. It follows the SRP by ensuring its only responsibility is to represent book details.

- **Library Class**: Manages a collection of `Book` objects, providing methods for adding and removing books.

- **BookFilter Interface**: Serves as a contract for all book filtering classes. This ensures that the filtering logic can be easily extended by implementing new filters.

- **AuthorFilter and TitleFilter Classes**: Provide specific filtering criteria by implementing the `BookFilter` interface. This design allows us to add new filtering criteria by simply extending the interface without modifying the existing codebase.

4. Code Walkthrough

The code consists of several components:

- **Book Class**: Represents a book with attributes like `title`, `author`, and `isbn`.

- **Library Class**: Manages a collection of books, providing methods for adding and removing books. Example usage:

```
  Library library = new Library();
  library.addBook(new Book("Effective Java", "Joshua Bloch", "1234567890"));
```

- **BookFilter Interface**: Defines a generic filter method for books, allowing for flexible extensions.

- **AuthorFilter and TitleFilter Classes**: Implement the `BookFilter` interface to provide filtering capabilities. Example usage:

```
  AuthorFilter authorFilter = new AuthorFilter("Joshua Bloch");
  List<Book> filteredBooks = LibrarySearch.filterBooks(library.getBooks(), authorFilter);
```

- **Main Class**: Contains the main method for executing the application and demonstrating the functionality.

5. Conclusion

In this laboratory work, the Single Responsibility Principle and Open/Closed Principle have been implemented in a simple Java library management system. Each class has a well-defined purpose, adhering to SRP, while OCP is demonstrated through the extensible filtering system. This design approach improves code maintainability and scalability, making it easier to add new features with minimal impact on existing code.