**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

Laboratory work 0:

SOLID

Elaborated:

st. gr. FAF-221 CalugareanuAna

Verified:

asist. univ. Furdul A.

Chişinău – 2024

**Objective:**

Implement 2 SOLID letters in a simple project.

**Theoretical Notes:**

SOLID Principles Overview:

The SOLID principles are a set of five guidelines in object-oriented programming that improve software design, making it more maintainable, scalable, and flexible.

1. Single Responsibility Principle (SRP):

A class should have only one responsibility. This ensures that each class is focused on a specific task, making the code easier to maintain and reducing the risk of unintended side effects when changes are made.

2. Open/Closed Principle (OCP):

Software entities should be open for extension but closed for modification. This allows developers to add new functionality without altering existing code, minimizing the risk of breaking established features.

3. Liskov Substitution Principle (LSP):

Subclasses should be replaceable with their base classes without affecting the correctness of the program. Following this principle ensures that objects of a derived class can be used wherever objects of the base class are expected, preserving system integrity.

4. Interface Segregation Principle (ISP):

Classes should not be forced to implement interfaces they don’t use. Instead, smaller, more specific interfaces should be created. This helps keep classes simple and focused on the methods they actually need, promoting cleaner code and better maintainability.

5. Dependency Inversion Principle (DIP):

High-level modules should not depend on low-level modules; both should depend on abstractions. This decouples different parts of the system, allowing components to be modified or replaced without affecting each other.

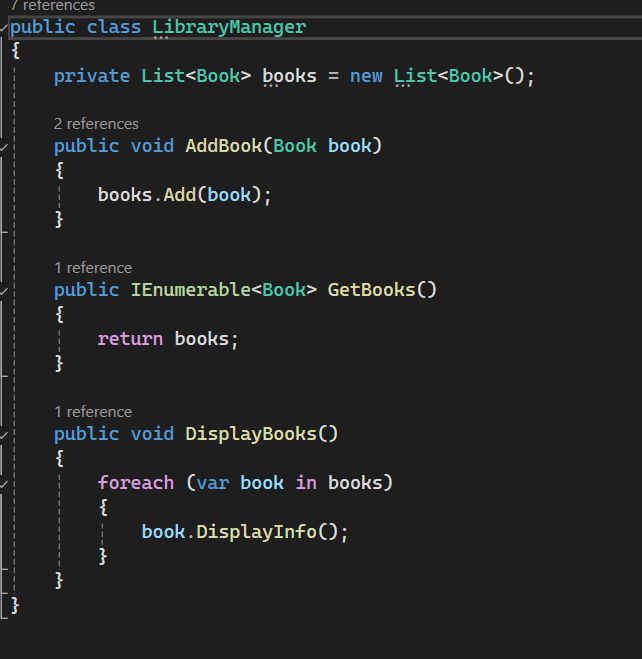
Importance of SOLID Principles:

The SOLID principles help create systems that are flexible and easier to maintain. By applying these principles, developers reduce dependencies, simplify code, and make it easier to add new features or make changes without breaking existing functionality. This leads to a codebase that is more scalable, reusable, and reliable in the long term.

**Implementation**

In this project, we applied two SOLID principles: Single Responsibility Principle (SRP) and Open/Closed Principle (OCP). These principles ensure the system is modular, easy to maintain, and extendable.  
 The application developed is a simple library management system that handles the addition and management of books, as well as loan processing. It follows two important SOLID principles: **Single Responsibility Principle (SRP)** and **Open/Closed Principle (OCP)**. The system uses the SRP by separating responsibilities into distinct classes: one for book management (LibraryManager) and one for processing loans (LoanProcessor). The OCP is demonstrated through the extensibility of the Book class, which allows the system to support different types of books, such as EBook and AudioBook, without modifying the core functionality.

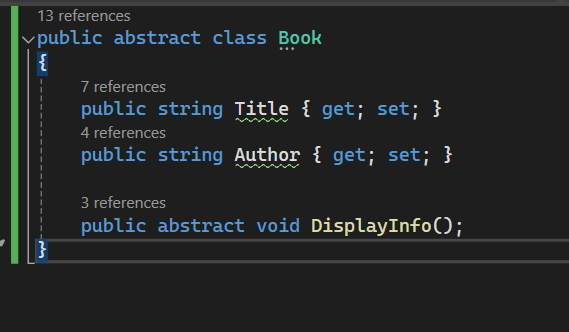
**Single Responsibility Principle (SRP)**:  
The system is designed with clear separation of responsibilities. The LibraryManager class handles all operations related to book management, such as adding and displaying books. The LoanProcessor class is responsible for loaning and returning books, ensuring each class focuses on a single task. This approach makes the system easier to maintain and reduces the chances of bugs caused by changes in unrelated areas.



**Figure 1.** Library Manager class

In this design, LibraryManager focus on distinct operations, following SRP by keeping unrelated responsibilities separate.

**Open/Closed Principle (OCP)**:  
The Book class serves as a base class that is closed for modification but open for extension. Specific types of books like EBook and AudioBook are added through inheritance, allowing the system to support new book types without changing the original class. This promotes flexibility, enabling future expansion while keeping the core logic intact.



**Figure 2.** Book class

By applying OCP, we ensure that new functionality (e.g., new book types) can be introduced through inheritance without modifying existing code. This makes the system more adaptable and reduces the risk of introducing bugs when extending the system.

**Conclusion**

The application of SOLID principles in this project has resulted in a flexible and maintainable system that adheres to modern software design practices. By implementing the **Single Responsibility Principle (SRP)**, we ensured that each class is focused on a single responsibility, reducing the complexity of the code and making it easier to maintain and debug. Separating book management from loan processing allows changes in one area to occur without affecting the other, improving modularity. Additionally, the **Open/Closed Principle (OCP)** allows the system to be extended with new features, like adding new book types, without modifying the existing code. This makes the codebase more robust to future changes and reduces the risk of introducing bugs during the development of new functionality.

The use of abstraction through inheritance demonstrates how easily new classes can be introduced while preserving the stability of the system. The flexibility provided by adhering to OCP ensures that the system can evolve with new requirements without compromising its core structure. Furthermore, following these principles makes the system more adaptable to new needs, minimizing future refactoring efforts. SOLID principles not only improve code quality but also provide a foundation for future growth. Overall, this approach has led to a scalable, extendable, and maintainable library management system that is ready for future expansion.