УДК[004.4’42:004.021](043.2)

**А. Ю. Білінський**

# ПРИНЦИПИ SOLID У РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вінницький національний технічний університет

***Анотація***

*У статті розглянуто фундаментальні принципи об’єктно-орієнтованого програмування, відомі під абревіатурою SOLID. Детально описано кожен із п’яти принципів: Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation та Dependency Inversion. Також показано практичне значення SOLID у сучасній розробці та продемонстровано приклади застосування підходів у популярних фреймворках, зокрема Spring. Дотримання SOLID сприяє підвищенню гнучкості, розширюваності та підтримуваності програмних систем.*

**Ключові слова:** SOLID, ООП, дизайн-принципи, розробка ПЗ, архітектура, Spring.

***Abstract***

*The article examines the fundamental object-oriented programming principles known as SOLID. Each of the five principles—Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion—is described in detail. The practical importance of SOLID in modern software development is highlighted, including examples in frameworks such as Spring. Applying these principles improves flexibility, extensibility, and maintainability of software systems.*

**Keywords:** SOLID, OOP, design principles, software development, architecture, Spring.

## Вступ

Принципи SOLID є одним із ключових наборів рекомендацій у сфері об’єктно-орієнтованої розробки. Вони були сформульовані Робертом Мартіном (Robert C. Martin) з метою підвищення якості архітектури програмних систем. SOLID забезпечує створення зрозумілого, гнучкого та легкого у підтримці коду, що особливо важливо при розробці великих вебдодатків, мікросервісних систем та корпоративних платформ.

До складу SOLID входять такі принципи:

S — Single Responsibility Principle (SRP) — принцип єдиної відповідальності.

O — Open/Closed Principle (OCP) — відкритості/закритості.

L — Liskov Substitution Principle (LSP) — підстановки Барбари Лісков.

I — Interface Segregation Principle (ISP) — розділення інтерфейсів.

D — Dependency Inversion Principle (DIP) — інверсії залежностей.

Метою роботи є розкриття кожного принципу та демонстрація їх практичного застосування в розробці ПЗ.

## Результати дослідження

1. Принцип єдиної відповідальності (SRP)

Кожен клас повинен мати лише одну причину для зміни. Це означає, що клас виконує тільки одну логічну функцію.

Переваги: спрощення структури коду, легше тестування, зниження зв’язності.

Приклад: у Spring сервіси та контролери не змішують бізнес-логіку та обробку HTTP-запитів — кожен компонент виконує одну роль.

2. Принцип відкритості/закритості (OCP)

Програмні сутності мають бути відкриті для розширення, але закриті для модифікації.

Суть: новий функціонал додається через успадкування або композицію, а не через зміну наявного коду.

Приклад: у Spring використання інтерфейсів сервісів дозволяє легко підміняти реалізації, не змінюючи код споживача.

3. Принцип підстановки Лісков (LSP)

Об’єкти підкласів повинні повністю заміняти об’єкти базового класу без порушення логіки.

Суть: успадкована поведінка не повинна суперечити контрактам батьківського класу.

Приклад: у системі ролей користувачів User → Admin мають працювати з однаковими інтерфейсами без винятків.

4. Принцип розділення інтерфейсів (ISP)

Багато спеціалізованих інтерфейсів кращі за один «масивний».

Переваги: зменшення залежностей, легке тестування, чіткі контракти між компонентами.

Приклад: замість одного великого інтерфейсу Repository можна мати PagingRepository, SortingRepository, CrudRepository.

5. Принцип інверсії залежностей (DIP)

Модулі верхнього рівня не повинні залежати від модулів нижнього рівня. Обидва типи мають залежати від абстракцій.

Суть: залежності будуються на інтерфейсах, а не на конкретних реалізаціях.

Приклад: Spring IoC Container впроваджує залежності через @Autowired, де сервіси інжектуються через абстракції.

На рисунку 1 зображено узагальнену схему принципів SOLID.

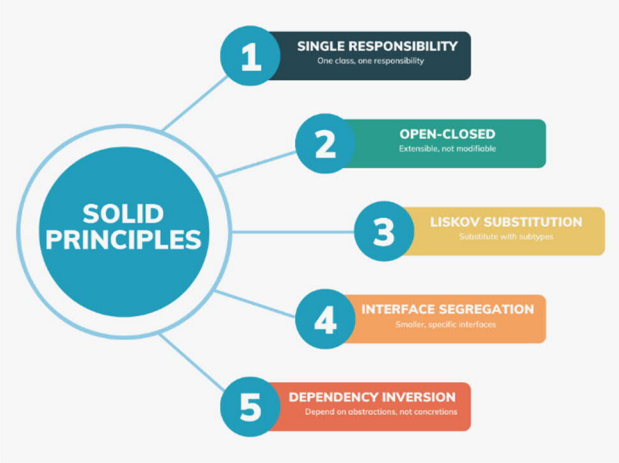


Рисунок 1 – Узагальнена схема принципів SOLID

На рисунку 1 [1] представлено узагальнену схему принципів SOLID, що визначають фундаментальні правила побудови якісної об’єктно-орієнтованої архітектури. Кожен із п’яти принципів — Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation та Dependency Inversion — спрямований на зменшення зв’язності, підвищення гнучкості та покращення підтримуваності програмного забезпечення. Дотримання SOLID забезпечує чіткий розподіл відповідальностей між компонентами, полегшує модифікацію системи та сприяє створенню масштабованих і надійних програмних рішень.

## Висновки

У роботі було розглянуто принципи SOLID, які є основою якісної об’єктно-орієнтованої розробки. Аналіз показав, що застосування SOLID зменшує зв’язність коду, покращує читабельність та полегшує масштабування програмних систем. Підхід широко використовується у сучасних фреймворках, зокрема Spring, що робить його актуальним для розробки вебдодатків та мікросервісних архітектур. Дотримання SOLID значно покращує життєвий цикл програмного продукту та підвищує якість програмного забезпечення.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Atıcı, B. (2023, 1 February). The SOLID Principles: A Beginner’s Guide to Writing Better Code. Medium. URL: https://birkanatici.medium.com/the-solid-principles-a-beginners-guide-to-writing-better-code-d882f78b59ca

2. Robert C. Martin. Clean Architecture. Pearson, 2019.

3. Robert C. Martin. Agile Software Development: Principles, Patterns, and Practices. Prentice Hall, 2002.

4. Офіційна документація Spring: https://spring.io/projects/spring-framework

***Артем Юрійович Білінський*** – студент ЗПІ-23Б факультету інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: bilinskiyartem110407@gmail.com;

***Artem Y. Bilinskyi*** – Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: bilinskiyartem110407@gmail.com;