

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
РАДІОЕЛЕКТРОНІКИ

Практична робота №1  
з дисципліни  
«Архітектура програмного забезпечення»  
на тему: «Патерни проєктування»

Виконав:

ст. гр. ПЗП-22-5

Коноваленко Даніїл Миколайович

Перевірив:

ст. викладач каф. ПІ

Сокорчук Ігор Петрович

Харків 2024

1 ІСТОРІЯ ЗМІН

Ім'я	Дата	Причина для змін	Версія
Коноваленко Даніїл	03.31.2025	Початкова версія	1.0

## 2 ЗАВДАННЯ

1. Підготувати доповідь на тему: «**Шаблон (патерн) проєктування ПЗ**»
2. Створити та оформити слайди презентації доповіді
3. Створити та опублікувати на YouTube відеозапис доповіді
4. Оформити та завантажити на [dl.nure.ua](https://dl.nure.ua) звіт до виконаної роботи
5. При потребі створити та налаштувати у GitHub обліковий запис для облікового запису студента в домені [@nure.ua](mailto:@nure.ua)
6. Створити для виконання завдань з дисципліни GitHub репозиторій із вказаним на <https://dl.nure.ua> іменем та з головною гілкою main
7. Створити у корені створеного репозиторію файл README.md з вказаним далі змістом та вказані далі директорії для окремих завдань
8. Експортувати звіт у файл у простому текстовому форматі та завантажити цей файл у директорію **Pract1** у репозиторії GitHub
9. Завантажити у вказану далі піддиректорію увесь програмний код, який розглядається у доповіді ПЗ1

### 3 ОПИС ВИКОНАНОЇ РОБОТИ

Під час підготовки доповіді-презентації виконано комплексний аналіз патернів проектування з акцентом на патерн Singleton як приклад структурного підходу до архітектури програмного забезпечення. Робота виконана у такій послідовності:

1. Вступний огляд патернів проектування

Проведено класифікацію патернів проектування на три основні категорії: креаційні, структурні та поведінкові. Визначено їх роль як перевірених часом архітектурних шаблонів, що спрощують розробку та підтримку програмного коду.

2. Детальний аналіз патерну Singleton

Патерн Singleton розглянуто як класичний приклад структурного патерну, що забезпечує існування єдиного екземпляра класу з контрольованим доступом до нього. Описано його ключові характеристики:

- забезпечення глобального стану;
  - обмеження створення кількох екземплярів об'єкта;
  - спрощення доступу до ресурсів.
- Наведено приклад базової реалізації Singleton у мові програмування JavaScript із використанням замикання (closure) для створення єдиного екземпляра.

3. Реалізація Singleton у Python

Представлено кілька підходів до реалізації патерну Singleton у Python:

- базова реалізація, що є простою, але не потокобезпечною;
- потокобезпечна реалізація з використанням об'єкта `threading.Lock`;

- спрощена реалізація з використанням декораторів або метакласів. Кожен підхід проаналізовано з точки зору його практичної застосовності та обмежень.

#### 4. Порівняння з альтернативними підходами

Проведено порівняння Singleton з іншими архітектурними рішеннями:

- Dependency Injection – сучасна альтернатива, що сприяє гнучкості та тестуванню коду;
- Factory – патерн для створення об'єктів без жорсткого обмеження кількості екземплярів;
- Multiton – розширення Singleton із підтримкою кількох іменованих екземплярів.

Визначено критерії вибору Singleton: необхідність глобального стану та контрольованого доступу.

#### 5. Практичні приклади використання

Описано типові сценарії застосування Singleton:

- логування у корпоративних системах;
- управління конфігураціями додатків;
- контроль з'єднань із базами даних;
- кешування даних;
- підтримка глобального стану додатку.

#### 6. Рекомендації та кращі практики

Сформульовано рекомендації щодо використання Singleton:

- застосовувати обережно, уникаючи надмірного використання;
- чітко визначати відповідальність об'єкта;

- розглядати альтернативи у випадках, коли глобальний стан не є критичним.

## **4 ВИСНОВКИ**

Патерн Singleton, як класичний приклад структурного підходу, відіграє важливу роль у ситуаціях, що вимагають єдиного екземпляра об'єкта з глобальним доступом. Проведений аналіз показав його переваги (контроль доступу, економія ресурсів) та недоліки (потенційні проблеми з тестуванням і потокобезпекою). Реалізація Singleton у Python демонструє гнучкість мови програмування, але вимагає врахування контексту використання. Порівняння з альтернативами, такими як Dependency Injection, підкреслює необхідність вибору оптимального рішення залежно від вимог проєкту.

## Додаток А

## Додаток Б



INTERACTIVE DESIGN  
VISUALIZATION

### Патерни проєктування: Глибоке занурення

В цій презентації будуть розглянуті ключові концепції, найпоширеніші архітектурні рішення, а також детально розглянемо патерн Singleton як класичний приклад структурного патерну. Мета – отримати практичні знання для покращення якості та ефективності програмного коду.

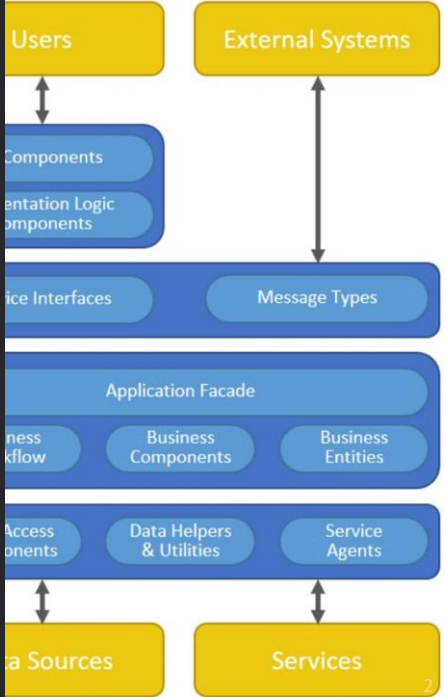
Автор: Данііл Коноваленко  
Група: ПЗПІ-22-5

Рисунок Б.1 – Вступний слайд

### Що таке патерни проєктування?

Патерни проєктування – це типові рішення поширених проблем у проєктуванні ПЗ. Вони є перевіреними часом архітектурними шаблонами, які спрощують розробку та підтримку коду. Патерни проєктування діляться на три основні категорії: креаційні, структурні та поведінкові. Розглянемо кожну з них детальніше.

- Типові рішення**  
Вирішення типових проблем у проєктуванні ПЗ
- Архітектурні шаблони**  
Перевірені часом шаблони
- Креаційні, структурні, поведінкові**  
Основні категорії патернів



2

Рисунок Б.2 – Загальне поняття патернів проєктування



## Singleton: Концепція та призначення

Патерн Singleton гарантує існування лише одного екземпляра класу в системі та надає глобальну точку доступу до нього. Він контролює створення та життєвий цикл об'єкта. Типові сценарії використання включають конфігурації, менеджери ресурсів, та інші глобально доступні об'єкти.

### Єдиний екземпляр

Гарантує єдиний екземпляр класу в системі

### Глобальна точка доступу

Глобальна точка доступу до унікального об'єкта

3

Рисунок Б.3 – Основне про Singleton

singleton pattern example

```
Singleton = (function () {  
    instance;  
  
    function createInstance() {  
        // Initialize your Singleton here  
        return {  
            // Methods and properties  
        };  
    }  
  
    return {  
        getInstance: function () {  
            if (!instance) {  
                instance = createInstance();  
            }  
            return instance;  
        }  
    };  
})();  
  
// Usage  
singletonInstance = Singleton.getInstance();
```

### Технічна реалізація Singleton

Для реалізації Singleton використовується приватний конструктор, який забороняє пряме створення екземплярів класу. Статичний метод **getInstance()** використовується для отримання екземпляра. Важливо враховувати потокобезпечність при реалізації, особливо в багатопоточних середовищах. Ініціалізація об'єкта може бути лінійною або ранньою.

**Приватний конструктор**  
Заборона прямого створення

**Статичний метод getInstance()  
getInstance()**  
Отримання екземпляра

**Потокобезпечність**  
Важливо для багатопоточних середовищ

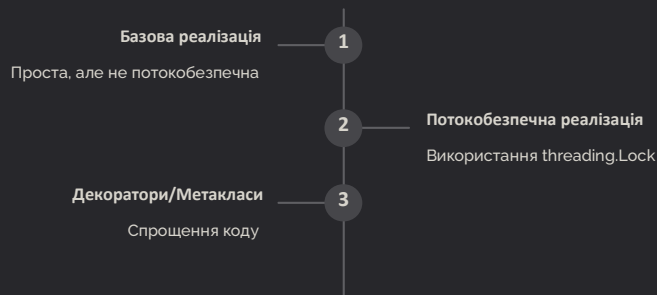
4

Рисунок Б.4 – Реалізація Singleton

```
# Define the aoi and input parameters of the workflow to run it.
aoi = workflow.get_example_aoi(as_dataframe=True)
input_parameters = workflow.construct_parameters(geometry=aoi,
                                                geometry_operation="bbox",
                                                start_date="2018-01-01",
                                                end_date="2020-12-31",
                                                limit=1)
```

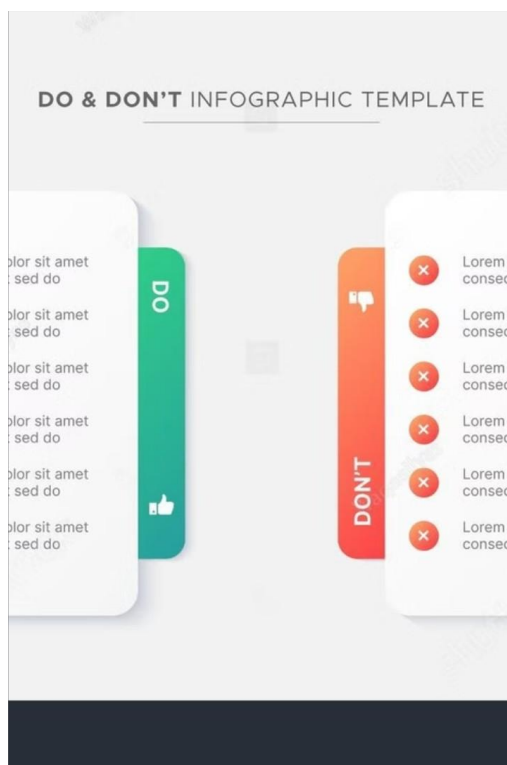
## Код реалізації Singleton на Python

Реалізація Singleton на Python може бути досягнута кількома способами. Базова реалізація проста, але не є потокобезпечною. Для потокобезпечної реалізації можна використовувати `threading.Lock` Також можна використовувати декоратори або метакласи для спрощення коду.



5

Рисунок Б.5 – Реалізація на Python



## Переваги патерну Singleton

Singleton забезпечує суворий контроль доступу до єдиного екземпляра, економить системні ресурси, сприяє глобальній координації в межах додатку та спрощує взаємодію між компонентами системи. Це робить його корисним у багатьох сценаріях, де потрібен єдиний об'єкт.



6

Рисунок Б.6 – Переваги Singleton



## Потенційні недоліки та обмеження

Singleton може порушувати принцип єдиної відповідальності, ускладнювати тестування, створювати приховані залежності між модулями та викликати проблеми з багатопоточністю. Важливо ретельно зважувати ці недоліки перед використанням патерну.

1

Порушення принципу

Єдиної відповідальності

2

Складність тестування

Ускладнення модульного тестування

3

Приховані залежності

Між модулями

4

Проблеми з багатопоточністю

Потребує синхронізації

7

Рисунок Б.7 – Мінуси та обмеження Singleton

## Альтернативи та контексти використання

Dependency Injection є сучасною альтернативою Singleton. Також важливо порівнювати Singleton з Factory та Multiton патернами. Критерії вибору Singleton включають необхідність глобального стану та контрольованого доступу. Кращі практики включають обережне використання та чітке визначення відповідальності.

Dependency Injection

Сучасна альтернатива

1

Factory та Multiton

Порівняння з іншими патернами

2

Кращі практики

Обережність та відповідальність

4

3

Критерії вибору

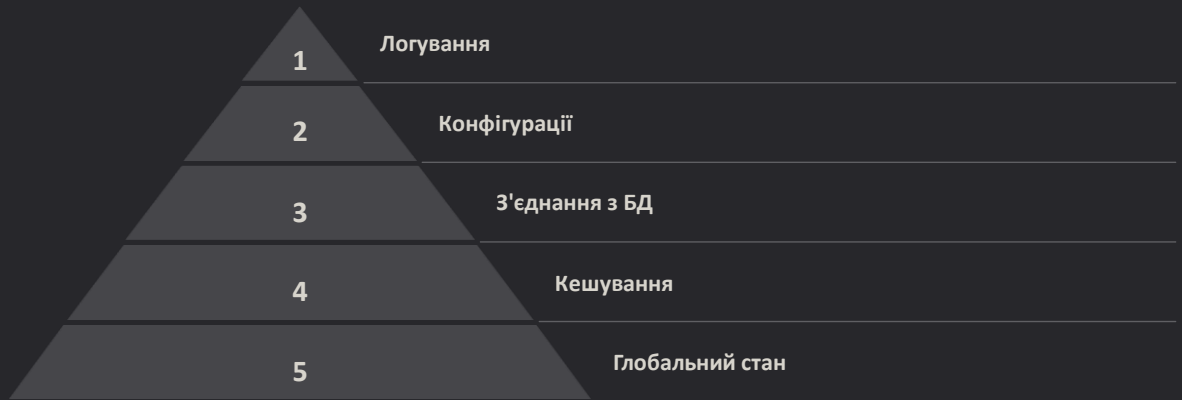
Глобальний стан та доступ

8

Рисунок Б.8 – Альтернативи патерну Singleton

## Реальні кейси використання

Singleton часто використовується для логування в корпоративних системах, налаштування конфігурацій додатків, управління з'єднаннями з базами даних, кешування та підтримки глобального стану додатку. Ці приклади демонструють практичну цінність патерну.



9

Рисунок Б.9 – Коли слід використовувати Singleton

## Висновки та перспективи

Патерни проєктування відіграють важливу роль в сучасній розробці. Singleton є класичним прикладом архітектурного рішення, але важливо розуміти контекст його використання. Постійне вдосконалення архітектурних підходів є ключем до створення якісного та підтримуваного програмного забезпечення.

1

Роль патернів

У сучасній розробці

2

Singleton

Класичний приклад

3

Контекст

Важливість розуміння



10

Рисунок Б.10 – Висновки

## Додаток В

### Приклад В.1

Базова реалізація Singleton у Python.

```
01 class Singleton:
02     _instance = None
03
04     def __new__(cls):
05         if cls._instance is None:
06             cls._instance = super().__new__(cls)
07             return cls._instance
08
09 obj1 = Singleton()
10 obj2 = Singleton()
11 print(obj1 is obj2)  # True
```

## Приклад В.2

Потокобезпечна реалізація Singleton у Python.

```
01 import threading
02
03 class Singleton:
04     _instance = None
05     _lock = threading.Lock()
06
07     def __new__(cls):
08         with cls._lock:
09             if cls._instance is None:
10                 cls._instance = super().__new__(cls)
11             return cls._instance
12
13 obj1 = Singleton()
14 obj2 = Singleton()
15 print(obj1 is obj2)    # True
```

## Приклад В.3

Реалізація Singleton із декоратором у Python:

```
01 def singleton(cls):
02     instances = {}
03     def get_instance(*args, **kwargs):
04         if cls not in instances:
05             instances[cls] = cls(*args, **kwargs)
06         return instances[cls]
07     return get_instance
08
09 @singleton
10 class Singleton:
11     pass
12
13 obj1 = Singleton()
14 obj2 = Singleton()
15 print(obj1 is obj2)  # True
```