**Назва лабораторної роботи:**

Аналіз та рефакторинг коду з виявленням запахів коду

**1. Обрана тема:**

**Аналіз коду**: Ідентифікація запахів коду (code smells), таких як дублювання коду, довгі методи, занадто багато обов’язків, та їх усунення за допомогою принципів рефакторингу і шаблонів проєктування.

**2. Мета лабораторної роботи:**

Навчитися виявляти запахи коду та ефективно усувати їх за допомогою рефакторингу і шаблонів проєктування з перевіркою змін модульними тестами.

**3. Завдання на виконання лабораторної роботи:**

* **Опис проблеми:** Наданий код містить типові «запахи коду», які ускладнюють його підтримку, повторне використання та масштабування.
* **Технічні вимоги:**
  + Використовувати мову Python 3.10+.
  + Виявити щонайменше 3 різні типи запахів коду.
  + Використати відповідні шаблони проєктування або принципи SOLID для їх усунення.
  + Написати не менше 10 модульних тестів (до і після рефакторингу).
* **Очікувані результати:**
  + Покращений, рефакторизований код.
  + Успішне проходження всіх модульних тестів.
  + Пояснення змін у звіті.

**4. Технічне завдання:**

* **Мова програмування:** Python 3.10
* **Фреймворк для тестування:** pytest 7.4
* **Додаткові бібліотеки:** flake8 (для аналізу стилю), pylint (для перевірки запахів)
* **IDE:** Visual Studio Code або PyCharm

**5. Приклад коду до та після рефакторингу:**

**До:**

# report\_before.py  
  
class Report:  
 def \_\_init\_\_(self, data):  
 self.data = data  
  
 def generate(self):  
 print("Generating report...")  
 for row in self.data:  
 print(f"Row: {row}")  
 print("Report complete.")  
  
 def save\_to\_file(self, filename):  
 with open(filename, 'w') as f:  
 f.write("Report:\n")  
 for row in self.data:  
 f.write(f"Row: {row}\n")  
 f.write("Report complete.\n")

**Запах: порушення принципу єдиного обов’язку (SRP)**

**Після:**

# report\_after.py  
  
class ReportGenerator:  
 def \_\_init\_\_(self, data, formatter):  
 self.data = data  
 self.formatter = formatter  
  
 def generate(self):  
 lines = ["Generating report..."]  
 for row in self.data:  
 lines.append(self.formatter.format(row))  
 lines.append("Report complete.")  
 return lines  
  
  
class SimpleFormatter:  
 def format(self, row):  
 return f"Row: {row}"  
  
  
class FileSaver:  
 def save(self, lines, filename):  
 with open(filename, 'w') as f:  
 for line in lines:  
 f.write(line + '\n')

Tests:

import os  
import pytest  
  
# До рефакторингу  
from prerefactor import Report  
  
# Після рефакторингу  
from postrefactor import ReportGenerator, SimpleFormatter, FileSaver  
  
  
# --- Тести до версії до рефакторингу ---  
def test\_generate\_report\_output(capsys):  
 r = Report(["apple", "banana", "cherry"])  
 r.generate()  
 captured = capsys.readouterr()  
 assert "Generating report..." in captured.out  
 assert "Row: apple" in captured.out  
 assert "Report complete." in captured.out  
  
def test\_save\_to\_file(tmp\_path):  
 file\_path = tmp\_path / "report.txt"  
 r = Report(["x", "y"])  
 r.save\_to\_file(file\_path)  
 content = file\_path.read\_text()  
 assert "Row: x" in content  
 assert "Report complete." in content  
  
  
# --- Тести до версії після рефакторингу ---  
def test\_report\_generator\_lines():  
 formatter = SimpleFormatter()  
 generator = ReportGenerator(["x", "y"], formatter)  
 lines = generator.generate()  
 assert lines[0] == "Generating report..."  
 assert lines[1] == "Row: x"  
 assert lines[2] == "Row: y"  
 assert lines[3] == "Report complete."  
  
def test\_file\_saver(tmp\_path):  
 saver = FileSaver()  
 lines = ["Line A", "Line B"]  
 file\_path = tmp\_path / "out.txt"  
 saver.save(lines, file\_path)  
 content = file\_path.read\_text()  
 assert "Line A" in content  
 assert "Line B" in content  
  
  
# --- Додаткові тести ---  
def test\_empty\_report\_before(capsys):  
 r = Report([])  
 r.generate()  
 captured = capsys.readouterr()  
 assert "Generating report..." in captured.out  
 assert "Report complete." in captured.out  
  
def test\_empty\_report\_after():  
 formatter = SimpleFormatter()  
 generator = ReportGenerator([], formatter)  
 lines = generator.generate()  
 assert lines == ["Generating report...", "Report complete."]  
  
def test\_formatter\_custom\_output():  
 class UpperFormatter:  
 def format(self, row):  
 return row.upper()  
  
 generator = ReportGenerator(["hello", "world"], UpperFormatter())  
 lines = generator.generate()  
 assert lines[1] == "HELLO"  
 assert lines[2] == "WORLD"  
  
def test\_file\_saver\_creates\_file(tmp\_path):  
 path = tmp\_path / "created.txt"  
 FileSaver().save(["Test line"], path)  
 assert path.exists()  
 assert path.read\_text().startswith("Test line")  
  
def test\_report\_output\_structure():  
 formatter = SimpleFormatter()  
 generator = ReportGenerator(["A", "B", "C"], formatter)  
 output = generator.generate()  
 assert output[0] == "Generating report..."  
 assert output[-1] == "Report complete."  
 assert len(output) == 5

**6. Критерії оцінювання:**

* **30%** — Дотримання технічного завдання (використання вказаних технологій, правильне оформлення).
* **20%** — Правильність виконання (функціональність збережена, код працює коректно).
* **20%** — Якість рефакторингу (усунення запахів коду, дотримання принципів SOLID або шаблонів проєктування).
* **30%** — Модульні тести (кількість ≥10, охоплення логіки, перевірка поведінки до та після змін).

**Висновки:**

Виконання лабораторної дозволило закріпити навички аналізу коду, практикувати виявлення недоліків та покращення структури ПЗ. Модульні тести допомогли впевнитися в збереженні функціональності після змін.