Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Звіт
про виконання лабораторної роботи №5
на тему:

«Об'єктно-орієнтований дизайн. Принципи SOLID»

Виконала:

Студентка групи ФеС-32

Філь Дарина

Львів - 2025

Мета роботи:

Ознайомлення з принципами об'єктно орієнтовно дизайну та застосування їх при рефакторингу коду.

Теоретичні відомості:

ООП (Об'єктно-орієнтоване програмування, OOP – Object-Oriented Programming)

ООП — це парадигма програмування, яка базується на використанні об'єктів.

Основні принципи ООП:

● Інкапсуляція – приховування деталей реалізації всередині

об'єкта.

● Наслідування – створення нових класів на основі існуючих.

● Поліморфізм – можливість взаємодії через загальні інтерфейси.

● Абстракція – виділення суттєвих характеристик без деталізації.

 ООП – це сам спосіб написання коду (наприклад, Java, Python, C++ підтримують ООП).

ООД (Об'єктно-орієнтоване проектування, OOD – Object-Oriented Design)

ООД – це методологія проектування програм, яка базується на ООП. Головне завдання – створення гнучкої та масштабованої архітектури.

✓ **ООП** ≈ Як писати код

✓ **ООД** ≈ Як правильно спроектувати систему

Для чого придумали SOLID?

SOLID – це набір принципів об'єктно-орієнтованого проектування (OOD), запропонований Робертом Мартіном (Uncle Bob). Його мета – створення чистого, підтримуваного та масштабованого коду.

S – Принцип єдиної відповідальності (SRP, Single Responsibility

Principle)

O – Принцип відкритості/закритості (OCP, Open/Closed Principle)

L – Принцип підстановки Барбари Ліскoв (LSP, Liskov Substitution

Principle)

I – Принцип розділення інтерфейсів (ISP, Interface Segregation

Principle)

D – Принцип інверсії залежностей (DIP, Dependency Inversion

Principle)

Хід роботи:

1. Використовуючи програмну реалізацію з лабораторної роботи 2 переписати даний код використовуючи SOLID принципи та об'єктно орієнтовану парадигму.

Абстрактні класи:

```python
from abc import ABC, abstractmethod

class Shape(ABC):  5 usages
    @abstractmethod  1 usage
    def area(self): pass

    @abstractmethod  1 usage
    def perimetr(self): pass

    def __str__(self):
        fig_name = self.__class__.__name__
        return f'{fig_name}: {self.__dict__}, prmtr = {self.perimetr()}, area = {self.area()}'

class SolidShape(Shape):  2 usages
    @abstractmethod  1 usage
    def volume(self): pass

    def __str__(self):
        return super().__str__() + f', volume = {self.volume()}'
```

CSV reader:

```python
import csv

class CSVReader:  2 usages
    def read_csv(self, file_name):  1 usage
        with open(file_name, 'r', encoding='utf8') as file:
            reader = csv.reader(file)
            return [[row[0], *map(int, row[1:])] for row in reader if row]
```

Figures:

```python
from abs import Shape, SolidShape
from managers import FigureManager
from math import sqrt

fm = FigureManager()

@fm.add_figure('circle')  4 usages
class Circle(Shape):
    def __init__(self, radius): self.radius = radius
    def area(self): return 3.14 * self.radius**2  1 usage
    def perimetr(self): return 2 * 3.14 * self.radius  1 usage

@fm.add_figure('triangle')  3 usages
class Triangle(Shape):
    def __init__(self, side): self.side = side
    def area(self): return (sqrt(3) / 4) * self.side**2  1 usage
    def perimetr(self): return self.side * 3  1 usage

@fm.add_figure('square')  3 usages
class Square(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self): return self.width * self.height  1 usage
    def perimetr(self): return 2 * (self.width + self.height)  1 usage
```

```python
@fm.add_figure('cube')  3 usages
class Cube(SolidShape):
    def __init__(self, side): self.side = side
    def area(self): return 6 * self.side**2  1 usage
    def perimetr(self): return 12 * self.side  1 usage
    def volume(self): return self.side**3  1 usage
```

Manager:

```python
class FigureManager:  4 usages
    def __init__(self):
        self.figures = {}

    def add_figure(self, name):  8 usages
        def decorator(cls):
            self.figures[name] = cls
            return cls
        return decorator

    def get_figure(self, name):  1 usage (1 dynamic)
        return self.figures.get(name)

class FigureCreator:  5 usages
    def __init__(self, fm):
        self.fm = fm

    def create_figure(self, figure_type, params):  3 usages
        fig_class = self.fm.get_figure(figure_type)
        if fig_class:
            return fig_class(*params)
        raise ValueError(f"Unknown figure type: {figure_type}")

class ProcessData:  2 usages
    def __init__(self, fc: FigureCreator):
        self.fc = fc

    def process_data(self, data):  1 usage
        return [self.fc.create_figure(fig_type, params) for fig_type, *params in data]
```

Tests:

```python
import unittest
from math import sqrt
from figures import Circle, Triangle, Square, Cube
from managers import FigureManager, FigureCreator

class TestShapes(unittest.TestCase):
    def test_circle(self):
        circle = Circle(5)
        self.assertAlmostEqual(circle.area(), 3.14 * 25)
        self.assertAlmostEqual(circle.perimetr(), 2 * 3.14 * 5)

    def test_triangle(self):
        triangle = Triangle(6)
        self.assertAlmostEqual(triangle.area(), (sqrt(3) / 4) * 36)
        self.assertEqual(triangle.perimetr(), second: 18)

    def test_square(self):
        square = Square( width: 4, height: 5)
        self.assertEqual(square.area(), second: 20)
        self.assertEqual(square.perimetr(), second: 18)

    def test_cube(self):
        cube = Cube(3)
        self.assertEqual(cube.area(), second: 54)
        self.assertEqual(cube.perimetr(), second: 36)
        self.assertEqual(cube.volume(), second: 27)
```

```python
class TestFigureCreator(unittest.TestCase):
    def setUp(self):
        self.fm = FigureManager()
        self.fm.add_figure('circle')(Circle)
        self.fm.add_figure('triangle')(Triangle)
        self.fm.add_figure('square')(Square)
        self.fm.add_figure('cube')(Cube)
        self.fc = FigureCreator(self.fm)

    def test_create_circle(self):
        self.assertIsInstance(self.fc.create_figure( figure_type: 'circle', params: [5]), Circle)

    def test_invalid_figure(self):
        with self.assertRaises(ValueError):
            self.fc.create_figure( figure_type: 'hexagon', params: [2])

if __name__ == "__main__":
    unittest.main()
```

Main:

```python
from figures import fm
from csv_tools import CSVReader
from managers import FigureCreator, ProcessData

if __name__ == "__main__":
    reader = CSVReader()
    fc = FigureCreator(fm)
    pd = ProcessData(fc)

    data = reader.read_csv('csv.csv')
    results = pd.process_data(data)

    for fig in results:
        print(fig)
```

2. Перевірити юніт тести зробити їх “зеленими”.

```
PS E:\СПП\Lab5> pytest tests.py
============================================================================= test session starts
platform win32 -- Python 3.12.8, pytest-8.3.5, pluggy-1.5.0
rootdir: E:\СПП\Lab5
collected 6 items

tests.py ......
============================================================================== 6 passed in 0.02s
```

3. Додати можливість порахувати об'єм просторових фігур (добавити приклад просторових фігур в csv файл).

```
circle,5
triangle,6
square,4,5
cube,3

circle, 7
```

```
PS E:\СПП\Lab5> python main.py
Circle: {'radius': 5}, prmtr = 31.400000000000002, area = 78.5
Triangle: {'side': 6}, prmtr = 18, area = 15.588457268119894
Square: {'width': 4, 'height': 5}, prmtr = 18, area = 20
Cube: {'side': 3}, prmtr = 36, area = 54, volume = 27
Circle: {'radius': 7}, prmtr = 43.96, area = 153.86
```

Висновок:

У цій лабораторній роботі, я ознайомилась з принципами об'єктно орієнтовно дизайну та застосував їх при рефакторингу коду.

Додаток:

abs.py

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self): pass

    @abstractmethod
    def perimetr(self): pass

    def __str__(self):
        fig_name = self.__class__.__name__
        return f'{fig_name}: {self.__dict__}, prmtr = {self.perimetr()}, area = {self.area()}'

class SolidShape(Shape):
    @abstractmethod
    def volume(self): pass

    def __str__(self):
        return super().__str__() + f', volume = {self.volume()}'
```

csv_tools.py

```
import csv

class CSVReader:
    def read_csv(self, file_name):
        with open(file_name, 'r', encoding='utf8') as file:
            reader = csv.reader(file)
            return [[row[0], *map(int, row[1:])] for row in reader if row]
```

figures.py

```
from abs import Shape, SolidShape
from managers import FigureManager
from math import sqrt

fm = FigureManager()

@fm.add_figure('circle')
class Circle(Shape):
    def __init__(self, radius): self.radius = radius
    def area(self): return 3.14 * self.radius**2
    def perimetr(self): return 2 * 3.14 * self.radius

@fm.add_figure('triangle')
class Triangle(Shape):
    def __init__(self, side): self.side = side
    def area(self): return (sqrt(3) / 4) * self.side**2
    def perimetr(self): return self.side * 3

@fm.add_figure('square')
class Square(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self): return self.width * self.height
    def perimetr(self): return 2 * (self.width + self.height)

@fm.add_figure('cube')
class Cube(SolidShape):
    def __init__(self, side): self.side = side
```

```
    def area(self): return 6 * self.side**2
    def perimetr(self): return 12 * self.side
    def volume(self): return self.side**3
```

managers.py

```
class FigureManager:
    def __init__(self):
        self.figures = {}

    def add_figure(self, name):
        def decorator(cls):
            self.figures[name] = cls
            return cls
        return decorator

    def get_figure(self, name):
        return self.figures.get(name)

class FigureCreator:
    def __init__(self, fm):
        self.fm = fm

    def create_figure(self, figure_type, params):
        fig_class = self.fm.get_figure(figure_type)
        if fig_class:
            return fig_class(*params)
        raise ValueError(f"Unknown figure type: {figure_type}")

class ProcessData:
    def __init__(self, fc: FigureCreator):
        self.fc = fc

    def process_data(self, data):
        return [self.fc.create_figure(fig_type, params) for fig_type, *params in data]
```

tests.py

```
import unittest
from math import sqrt
from figures import Circle, Triangle, Square, Cube
from managers import FigureManager, FigureCreator

class TestShapes(unittest.TestCase):
    def test_circle(self):
        circle = Circle(5)
        self.assertAlmostEqual(circle.area(), 3.14 * 25)
        self.assertAlmostEqual(circle.perimetr(), 2 * 3.14 * 5)

    def test_triangle(self):
        triangle = Triangle(6)
        self.assertAlmostEqual(triangle.area(), (sqrt(3) / 4) * 36)
        self.assertEqual(triangle.perimetr(), 18)

    def test_square(self):
        square = Square(4, 5)
        self.assertEqual(square.area(), 20)
        self.assertEqual(square.perimetr(), 18)

    def test_cube(self):
        cube = Cube(3)
        self.assertEqual(cube.area(), 54)
        self.assertEqual(cube.perimetr(), 36)
        self.assertEqual(cube.volume(), 27)

class TestFigureCreator(unittest.TestCase):
    def setUp(self):
        self.fm = FigureManager()
        self.fm.add_figure('circle')(Circle)
        self.fm.add_figure('triangle')(Triangle)
        self.fm.add_figure('square')(Square)
        self.fm.add_figure('cube')(Cube)
        self.fc = FigureCreator(self.fm)

    def test_create_circle(self):
        self.assertIsInstance(self.fc.create_figure('circle', [5]), Circle)
```

```
    def test_invalid_figure(self):
        with self.assertRaises(ValueError):
            self.fc.create_figure('hexagon', [2])

if __name__ == "__main__":
    unittest.main()
```

main.py

```
from figures import fm
from csv_tools import CSVReader
from managers import FigureCreator, ProcessData

if __name__ == "__main__":
    reader = CSVReader()
    fc = FigureCreator(fm)
    pd = ProcessData(fc)

    data = reader.read_csv('csv.csv')
    results = pd.process_data(data)

    for fig in results:
        print(fig)
```