Лабораторна робота 8

Тема: Основи ООП: інкапсуляція, успадкування та поліморфізм.

Мета: Навчитися керувати станом незалежних об’єктів і викликати перевизначені методи через спільний інтерфейс.

Зміст роботи

Завдання 1. Прямокутник із контрольованим станом

Створіть клас Rectangle з внутрішніми атрибутами _width, _height. Розміри — додатні скінченні числа; перевірку виконайте під час створення та зміни.

1. Реалізуйте area(), perimeter() і dimensions(), що повертають площу, периметр та кортеж розмірів.
2. Реалізуйте set_dimensions(width, height). Перевірте обидва аргументи до зміни стану; за недопустимих даних підніміть ValueError, зберігши попередні розміри.
3. У головній програмі створіть два об’єкти, змініть лише перший та виведіть показники обох. Обробіть один невдалий виклик методу й покажіть збереження стану.

Методи повертають результати, а введення, виведення та обробку ValueError виконує головна програма. Початкові розміри можна задати в коді. Перевірте власні дані. Для скінченності використайте math.isfinite(). Дії з двома об’єктами в таблиці 1 виконуйте послідовно; перевірку конструктора — окремо. Позначення inf і nan означають float("inf") і float("nan").

Таблиця 1 — Контрольна перевірка стану прямокутників

Дія	Стан і показники	Що перевіряється
Створити перший (3, 4), другий (2, 5).	Перший: розміри (3, 4), площа 12, периметр 14. Другий: розміри (2, 5), площа 10, периметр 14.	Обчислення й два об’єкти
Змінити перший на (6, 4).	Перший: розміри (6, 4), площа 24, периметр 20. Другий: (2, 5), площа 10, периметр 14.	Незалежність стану

Дія	Стан і показники	Що перевіряється
Спробувати змінити перший на (7, 0), (-1, 4), (inf, 4), (7, nan).	Кожна спроба піднімає ValueError. Перший зберігає розміри (6, 4), площу 24 і периметр 20; другий не змінюється.	Перевірки до присвоєння; межі й скінченність
Спробувати створити об’єкт із (0, 4).	ValueError; коректний об’єкт із такими розмірами не створюється.	Перевірка під час створення

Завдання 2. Індивідуальний клас колекції

Створіть клас для свого варіанта таблиці 3. Об’єкт має назву, власний внутрішній список _values та, за потреби, власний параметр. Введення й виведення виконуйте лише в головній програмі.

1. У __init__ створіть новий порожній список для кожного об’єкта. Реалізуйте add(value), clear() і values(), який повертає незалежну копію поточних даних.
2. Реалізуйте result() та summary() за правилом варіанта. result() щоразу повертає новий список; обидва методи не змінюють _values. clear() очищує лише дані об’єкта, зберігаючи назву й параметр.
3. Створіть два об’єкти, додайте різні дані, виведіть результати. Додайте елемент і очистіть лише перший; покажіть незалежність другого.
4. Змініть списки, повернені values() та result(), і переконайтеся, що дані й повторний результат об’єкта не змінилися.

Дані й параметри коректні. Числові варіанти працюють із цілими числами, словесні — з непорожніми словами; слова та суфікс переводьте в нижній регістр. Параметр k є цілим. Порядок елементів зберігайте, якщо варіант явно не вимагає іншого.

Для параметризованих варіантів задайте двом об’єктам різні параметри й доберіть дані, на яких видно їхній вплив. Перевірте повторення, відсутність результатів, порожній об’єкт і власні дані; значення summary() для порожнього випадку наведено в таблиці 3.

У таблиці 3 «Вхід» — дані, послідовно додані до першого об’єкта; «Вивід» — повернені значення result() і summary(), які друкує головна програма. Другий об’єкт наповніть власними даними.

Таблиця 2 — Спільна контрольна перевірка колекції

Дія	**Очікувана поведінка**	**Що перевіряється**
Створити два об’єкти. До першого додати дані з таблиці 3, до другого — інші.	Перший повертає контрольні result() і summary(); другий працює зі своїми даними.	Окремий стан
Зберегти values() другого. Додати елемент лише до першого.	Дані першого доповнено. values(), result() і summary() другого не змінюються.	Незалежність add()
До списку з values() додати елемент; список із result() очистити.	Внутрішні дані й повторний result() об’єкта залишаються такими, як до зовнішніх змін.	Незалежні повернені списки
Очистити перший через clear().	Його values() і result() — []; summary() — за варіантом. Назва й параметр збережені; другий об’єкт незмінний.	clear() і порожній стан
Для параметризованого варіанта задати різні параметри та дані, на яких видно різницю.	Кожен об’єкт використовує власний параметр; зміна даних першого не змінює налаштування другого.	Незалежність параметрів

Таблиця 3 — Індивідуальні класи колекцій

Варіант	**Дані й параметри**	**Результат result()**	**Підсумок summary()**	**Контрольний приклад**
1	Цілі числа.	унікальні в порядку появи	(кількість, сума) унікальних, для порожніх (0, 0).	**Вхід:** 2 1 2 3 **Вивід:** result() = [2, 1, 3] summary() = (3, 6)
2	Цілі числа.	стиснення сусідніх повторень	довжина найдовшої серії збережених даних; для порожніх 0.	**Вхід:** 2 2 1 1 1 2 **Вивід:** result() = [2, 1, 2] summary() = 3
3	Цілі числа.	суми префіксів	найбільша сума префікса, для порожніх 0.	**Вхід:** 2 -5 4 **Вивід:** result() = [2, -3, 1] summary() = 2
4	Цілі числа.	значення з частотою рівно 1	їхня сума, для порожніх 0.	**Вхід:** 2 1 2 3 **Вивід:** result() = [1, 3] summary() = 4
5	Цілі числа.	різниці сусідніх елементів	кількість додатних різниць, для порожніх 0.	**Вхід:** 3 5 5 2 **Вивід:** result() = [2, 0, -3] summary() = 1

Варіант	Дані й параметри	Результат result()	Підсумок summary()	Контрольний приклад
6	Слова.	унікальні слова, упорядковані через sorted()	сума їхніх довжин, для порожніх 0.	**Вхід:** код я код мова **Вивід:** result() = ['код', 'мова', 'я'] summary() = 8
7	Слова. параметр — непорожній суфікс	слова із суфіксом	(кількість, сума довжин), для порожніх (0, 0).	**Вхід:** мова книга код; суфікс а **Вивід:** result() = ['мова', 'книга'] summary() = (2, 9)
8	Цілі числа. k — ціле > 0	перші k унікальних значень	їхня сума, для порожніх 0.	**Вхід:** 2 1 2 3; k = 2 **Вивід:** result() = [2, 1] summary() = 3
9	Цілі числа.	значення, більші за середнє початкових	їхня кількість, для порожніх 0.	**Вхід:** 1 2 6 **Вивід:** result() = [6] summary() = 1
10	Слова.	слова, що не повторюються	перше в початковому порядку найдовше слово з result(); якщо таких слів немає — None.	**Вхід:** код мова код дані **Вивід:** result() = ['мова', 'дані'] summary() = 'мова'
11	Цілі числа.	пари (значення, довжина) для кожної серії однакових сусідніх значень	кількість серій, для порожніх 0.	**Вхід:** 1 1 3 3 3 **Вивід:** result() = [(1, 2), (3, 3)] summary() = 2
12	Слова. k — ціле ≥ 0	унікальні слова довжиною > k у порядку появи	сума довжин, для порожніх 0.	**Вхід:** код мова мова python; k = 3 **Вивід:** result() = ['мова', 'python'] summary() = 10
13	Цілі числа. k — ціле > 0	суми послідовних блоків по k елементів; останній може бути коротшим.	кількість блоків, для порожніх 0.	**Вхід:** 1 2 3 4 5; k = 2 **Вивід:** result() = [3, 7, 5] summary() = 3
14	Цілі числа.	внутрішні елементи, строго більші за обох сусідів; крайні не перевіряти	їхня сума, для порожніх 0.	**Вхід:** 1 4 2 5 3 **Вивід:** result() = [4, 5] summary() = 9
15	Слова.	слова, довші за безпосередньо попереднє; перше не перевіряти	кількість таких слів, для порожніх 0.	**Вхід:** я код мова кіт **Вивід:** result() = ['код', 'мова'] summary() = 2

Завдання 3. Успадкування й поліморфні виклики

Створіть базовий клас Figure з назвою та методом area(), який піднімає NotImplementedError. Реалізуйте два похідні класи RectangleFigure й Circle з додатними розмірами; використайте super().__init__() для назви.

1. Перевизначте area() у кожному похідному класі: площа прямокутника — width × height, круга — πr². Для π використайте math.pi.
2. Створіть список щонайменше з трьох фігур обох типів. В одному циклі виведіть назви й площі з двома десятковими знаками та загальну площу. Не використовуйте перевірки типу для вибору формули.

Дані для завдання 3 коректні: розміри — додатні скінченні числа. Назви фігур задайте під час створення. У таблиці 4 пари означають (ширина, висота), r — радіус. Перевірте власні розміри. Figure не використовуйте як готову фігуру; множинне успадкування й абстрактні класи не потрібні.

Таблиця 4 — Контрольна перевірка поліморфних викликів

Фігури та розміри	**Очікуване виведення**	**Що перевіряється**
Прямокутник 1 — (3, 4). Круг — r = 1. Прямокутник 2 — (2, 5).	Прямокутник 1: 12.00 Круг: 3.14 Прямокутник 2: 10.00 Загальна площа: 25.14	Один цикл викликає area() для обох типів
Прямокутник 1 — (1, 2). Круг — r = 2. Прямокутник 2 — (2, 3).	Прямокутник 1: 2.00 Круг: 12.57 Прямокутник 2: 6.00 Загальна площа: 20.57	Інші розміри; сума без проміжного округлення

Оформлення звіту

Звіт оформлюють за чинними вимогами курсу (документ «Вимоги до оформлення»).

Для кожного завдання у звіті подають формулювання, текст програми та скріншот результату в консолі.

Методичні рекомендації

Клас, екземпляр та інкапсуляція

class визначає тип об’єктів. __init__ налаштовує стан екземпляра; self посилається на поточний об’єкт.

Присвоєння self._values = [] усередині __init__ створює окремий список для кожного екземпляра. Список як атрибут класу був би спільним.

Інкапсуляція зосереджує дані та правила зміни стану в одному класі. Початковий підкреслювач _width означає домовленість про внутрішнє використання, а не технічну заборону доступу. Зовнішній код звертається до методів, які забезпечують допустимий стан.

Метод зміни спочатку перевіряє всі нові значення, а лише потім виконує присвоєння. Якщо друге значення неправильне, перше також не повинно змінитися. raise ValueError("опис") повідомляє про недопустимий стан; головна програма може його обробити через except ValueError. math.isfinite() не допускає нескінченність або NaN.

self посилається на окремий екземпляр

Код Python

```
class Counter:
    def __init__(self, start):
        self._value = start

    def increment(self):
        self._value += 1

    def value(self):
        return self._value

first = Counter(2)
second = Counter(7)
first.increment()
print(first.value(), second.value())
```

Результат виконання

```
3 7
```

Кожен __init__ установив власний атрибут. increment() змінив тільки об’єкт first. Підкреслювач є домовленістю про внутрішній стан; зовнішній код тут використовує метод value().

Перевірка всіх аргументів перед зміною стану

Код Python

```
class Interval:
    def __init__(self, left, right):
        self.set_limits(left, right)

    def set_limits(self, left, right):
        if left >= right:
            raise ValueError("Ліва межа має бути меншою")
        self._left = left
        self._right = right

    def limits(self):
        return self._left, self._right

interval = Interval(1, 5)
try:
    interval.set_limits(9, 2)
except ValueError:
    print("Зміну відхилено")
print(interval.limits())
```

Результат виконання

```
Зміну відхилено
(1, 5)
```

Умова перевіряється до обох присвоєнь. Невдалий виклик зберігає попередні межі, а конструктор використовує ту саму перевірку. У Rectangle правило інше: обидва розміри мають бути додатними й скінченними.

Додатність і скінченність числа

Для розміру потрібні дві умови одночасно: число скінченне й більше за нуль. Саме перевірка value > 0 не відхиляє додатну нескінченність.

Код Python

```
import math

for value in [2.5, 0, float("nan"), float("inf")]:
    valid = math.isfinite(value) and value > 0
    print(value, valid)
```

Результат виконання

```
2.5 True
0 False
nan False
inf False
```

math.isfinite() відхиляє nan та inf, а порівняння з нулем — нуль і від’ємні числа. У методі зміни Rectangle перевірте обидва аргументи до присвоєння атрибутів; для недопустимих розмірів підніміть ValueError.

Колекція як незалежний стан об’єкта

Кожен об’єкт зберігає дані й налаштування у власних атрибутах. add() змінює тільки self._values, clear() очищує його. Для простого списку чисел або рядків return self._values.copy() достатньо, щоб зовнішнє append() не змінювало внутрішній список.

result() створює новий список на кожен виклик. Не зберігайте в ньому посилання на внутрішні змінні списки. У цих варіантах результати містять числа, рядки або кортежі чисел; зміна зовнішнього списку не змінить елементи об’єкта. summary() повертає визначений умовою показник.

Копія відділяє зовнішні зміни від стану

Код Python

```
class Basket:
    def __init__(self):
        self._items = []

    def add(self, item):
        self._items.append(item)

    def values(self):
        return self._items.copy()

first = Basket()
second = Basket()
first.add("код")
outside = first.values()
outside.append("зайве")
print(first.values())
print(second.values())
```

Результат виконання

```
['код']
[]
```

Внутрішній список створюється в __init__, а не в тілі класу. Додавання до outside не змінило first. У цих задачах елементи — незмінні числа, рядки або кортежі, тому копії зовнішнього списку достатньо.

Дані для обчислень беріть із self._values. Унікальність означає відсутність повторів у результаті; частота — кількість поява значення в усіх збережених даних. Серія — послідовні однакові сусідні значення, а префікс — початкова частина списку від першого елемента.

Для блоків по k елементів перебирайте range(0, len(values), k) і зріз values[start:start + k]. Останній коротший блок теж враховуйте.

Порівняння з попереднім починайте з індексу 1; для двох сусідів використовуйте індекси від 1 до len(values) − 2.

Не пропустіть останню серію. Перед обчисленням середнього перевірте, чи є дані. Максимум непорожньої послідовності може бути від’ємним, тому не підмінюйте його початковим нулем. Для першого найдовшого слова оновлюйте вибір лише за строго більшої довжини; None означає, що відповідного слова немає.

Суми префіксів і від’ємний максимум

Накопичувана сума після кожного елемента утворює новий елемент результату. Приклад показує сам прийом; у методі класу джерелом буде self._values.

Код Python

```
values = [-3, 1, -2]
prefix_sums = []
total = 0
for value in values:
    total += value
    prefix_sums.append(total)
print(prefix_sums)
print(max(prefix_sums) if prefix_sums else 0)
```

Результат виконання

```
[-3, -2, -4]
-2
```

Початковий список не змінюється. Найбільша сума тут −2, тому нуль був би неправильним максимумом. Для порожнього результату умовний вираз повертає 0.

Серії однакових сусідніх значень

Однакові числа, розділені іншими числами, утворюють різні серії. Кожну серію збережемо як кортеж (значення, довжина).

Код Python

```
values = [4, 4, 1, 4, 4, 4]
runs = []
for value in values:
    if not runs or value != runs[-1][0]:
        runs.append((value, 1))
    else:
        runs[-1] = (value, runs[-1][1] + 1)
print(runs)
print(max(length for value, length in runs) if runs else 0)
```

Результат виконання

```
[(4, 2), (1, 1), (4, 3)]
3
```

Перші й останні четвірки належать до різних серій довжиною 2 та 3. Кортеж незмінний, тому останній елемент списку замінюється новим кортежем. Кожне повторення одразу збільшує довжину; остання серія теж потрапляє в результат.

Успадкування та перевизначення

class Child(Base) успадковує поведінку Base. super().__init__(name) викликає ініціалізацію базового класу для поточного об’єкта. Похідний клас може оголосити метод із тим самим іменем — це перевизначення. NotImplementedError у базовому методі пояснює, що конкретну поведінку має дати нащадок.

Поліморфізм дозволяє викликати object.area() однаково для різних фігур. Python обирає реалізацію за фактичним об’єктом; if isinstance(...) для вибору формули в циклі не потрібний. math.pi з модуля math — наближене число π, a r ** 2 — квадрат радіуса.

Класи Rectangle із завдання 1 та RectangleFigure із завдання 3 можна реалізувати окремо: перший демонструє контроль змін стану, другий — спільний інтерфейс фігур. Властивості property та складні ієрархії в цій роботі не обов’язкові.

super() викликає ініціалізацію базового класу

Код Python

```
class Named:
    def __init__(self, name):
        self.name = name

class Tagged(Named):
    def __init__(self, name, tag):
        super().__init__(name)
        self.tag = tag

item = Tagged("Мова", "Python")
print(item.name, item.tag)
```

Результат виконання

```
Мова Python
```

Похідний клас повторно використав установлення name, а tag додав сам. super() працює з поточним об’єктом; це не створення окремого базового екземпляра.

Один виклик, різні реалізації методу

Код Python

```
class Greeting:
    def message(self):
        raise NotImplementedError

class UkrainianGreeting(Greeting):
    def message(self):
        return "Вітаю"

class EnglishGreeting(Greeting):
    def message(self):
        return "Hello"

for greeting in [UkrainianGreeting(), EnglishGreeting()]:
    print(greeting.message())
```

Результат виконання

```
Вітаю
Hello
```

Похідні класи перевизначили message(). Цикл викликає той самий метод без перевірок типу, що демонструє поліморфізм. У завданні таким спільним методом є area().

Часті помилки

Не створюйте спільний список даних у тілі класу й не повертайте внутрішній список напряму.

Перевіряйте всі аргументи до зміни стану. Не забувайте self і return. У поліморфному циклі викликайте спільний метод без перевірок типу.

Приклади виконання завдань

Приклад до завдання 1

Клас Timer зберігає невід’ємну цілу кількість секунд і повертає пару (хвилини, секунди). Метод зміни перевіряє нове значення до присвоєння. Створимо два незалежні таймери, змінимо перший і перевіримо невдалу спробу зміни.

Код Python

```python
class Timer:
    def __init__(self, seconds):
        self.set_seconds(seconds)

    def set_seconds(self, seconds):
        if type(seconds) is not int or seconds < 0:
            raise ValueError("Секунди мають бути невід’ємним цілим")
        self._seconds = seconds

    def minutes_and_seconds(self):
        return self._seconds // 60, self._seconds % 60

first = Timer(75)
second = Timer(130)
print(f"Спочатку: {first.minutes_and_seconds()}; {second.minutes_and_seconds()}")
first.set_seconds(90)
try:
    first.set_seconds(-1)
except ValueError:
    print("Зміну відхилено")
print(f"Після зміни: {first.minutes_and_seconds()}; {second.minutes_and_seconds()}")
```

Приклад виконання в консолі

```
Спочатку: (1, 15); (2, 10)
Зміну відхилено
Після зміни: (1, 30); (2, 10)
```

Пояснення до прикладу

__init__ викликає set_seconds(), тому створення й подальша зміна мають однакову перевірку. type(seconds) is int відхиляє дробові значення та True/False; порівняння з нулем відхиляє від’ємні числа.

Для 75 секунд операції // 60 і % 60 дають (1, 15), для 130 — (2, 10). Після set_seconds(90) перший таймер повертає (1, 30).

Виклик set_seconds(-1) піднімає ValueError до присвоєння: зберігаються 90 секунд. Другий таймер залишається зі своїми 130 секундами. У Rectangle так само перевірте обидва розміри до зміни стану.

Приклад до завдання 2

Клас WordShelf зберігає слова й власну довжину початку слова. result() повертає до заданої кількості перших символів кожного слова, summary() — загальну кількість символів усіх збережених слів. Два об’єкти матимуть різні параметри 2 і 3; параметри — коректні додатні цілі числа.

Код Python

```
class WordShelf:
    def __init__(self, name, prefix_length):
        self.name = name
        self._prefix_length = prefix_length
        self._words = []

    def add(self, word):
        self._words.append(word.lower())

    def clear(self):
        self._words.clear()

    def values(self):
        return self._words.copy()

    def result(self):
        return [word[:self._prefix_length] for word in self._words]

    def summary(self):
        return sum(len(word) for word in self._words)

first = WordShelf("Перший", 2)
second = WordShelf("Другий", 3)
for word in input("Слова першого: ").split():
    first.add(word)
for word in input("Слова другого: ").split():
    second.add(word)
outside = first.values()
outside.append("зайве")
prefixes = first.result()
prefixes.clear()
print(f"Перший, дані: {first.values()}")
print(f"Початки: {first.result()}; символів: {first.summary()}")
first.add("тест")
print(f"Після add(), перший: {first.values()}")
print(f"Початки: {first.result()}; символів: {first.summary()}")
print(f"Другий: {second.values()}; початки: {second.result()}")
first.clear()
print(f"Після clear(), перший: {first.values()}")
print(f"Результат: {first.result()}; символів: {first.summary()}")
print(f"Другий: {second.values()}; початки: {second.result()}")
```

Приклад виконання в консолі

```
Слова першого: КОД мова
Слова другого: дані
Перший, дані: ['код', 'мова']
Початки: ['ко', 'мо']; символів: 7
Після add(), перший: ['код', 'мова', 'тест']
Початки: ['ко', 'мо', 'те']; символів: 11
Другий: ['дані']; початки: ['дан']
Після clear(), перший: []
Результат: []; символів: 0
Другий: ['дані']; початки: ['дан']
```

Пояснення до прикладу

Кожен об’єкт має власні _words і _prefix_length. Перший повертає двосимвольні початки «ко», «мо», другий — трисимвольний «дан». lower() нормалізує введене «КОД» до «код».

values() повертає копію, а result() створює новий список. Тому додавання «зайве» до outside та очищення prefixes не змінюють дані першого об’єкта й повторний результат. Початковий summary() дорівнює 3 + 4 = 7.

Після add("тест") лише перший об’єкт має три слова й summary() = 11. clear() дає для нього порожні дані, порожній результат і підсумок 0; назва та параметр зберігаються. Дані другого й його трисимвольні початки залишаються незмінними.

Приклад до завдання 3

Базовий клас Delivery зберігає назву й оголошує cost(). LocalDelivery рахує оплату як 20 + 5 × відстань, ParcelDelivery — як 30 + 10 × маса. Вхідні значення коректні й невід’ємні. Один цикл викликає cost() для обох типів і підсумовує оплату.

Код Python

```python
class Delivery:
    def __init__(self, name):
        self.name = name

    def cost(self):
        raise NotImplementedError("Потрібен конкретний тип доставки")

class LocalDelivery(Delivery):
    def __init__(self, name, distance):
        super().__init__(name)
        self.distance = distance

    def cost(self):
        return 20 + 5 * self.distance

class ParcelDelivery(Delivery):
    def __init__(self, name, mass):
        super().__init__(name)
        self.mass = mass

    def cost(self):
        return 30 + 10 * self.mass

deliveries = [
    LocalDelivery("Місто", 3),
    ParcelDelivery("Посилка", 2),
    LocalDelivery("Поруч", 1),
]
total = 0
for delivery in deliveries:
    value = delivery.cost()
    print(f"{delivery.name}: {value:.2f}")
    total += value
print(f"Разом: {total:.2f}")
```

Приклад виконання в консолі

```
Місто: 35.00
Посилка: 50.00
Поруч: 25.00
Разом: 110.00
```

Пояснення до прикладу

super().__init__(name) установлює назву в поточному об’єкті, а кожен нащадок додає свій параметр: відстань або масу. Метод cost() базового класу піднімає NotImplementedError; конкретні обчислення реалізовано в нащадках.

Для «Місто» отримуємо 20 + 5 × 3 = 35, для «Посилка» — 30 + 10 × 2 = 50, для «Поруч» — 20 + 5 × 1 = 25. Сума дорівнює 110.00; формат .2f задає два десяткові знаки під час виведення.

Цикл викликає delivery.cost() без перевірок типу. У завданні 3 так само викликайте area() й підсумовуйте повернені площі без проміжного округлення.

Контрольні питання

1. Чим клас відрізняється від екземпляра?
2. Що означає self?
3. Яке призначення __init__?
4. Що означає один підкреслювач у назві атрибута?
5. Чому всі перевірки виконують до зміни стану?
6. Як створити незалежні списки для двох об’єктів?
7. Чому повернення внутрішнього списку небезпечне?
8. Як працює super().__init__()?
9. Що таке перевизначення методу?
10. Як один цикл демонструє поліморфізм?