Міністерство освіти і науки України
Державний університет “Житомирська політехніка”

Кафедра інженерії програмного забезпечення
Група: ВТ-21-1[1]

Програмування мовою Python
Лабораторна робота № 9
«КЛАСИ. Ч. 3»

Виконав: Бабушко А. С.
Прийняв: Морозов Д. С.

					«Житомирська політехніка».22.121.01.000–Лр9		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Бабушко А.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 16
Перевір.		Морозов Д.С.					
Керівник						ФІКТ Гр. ВТ-21-1[1]	
Н. контр.							
Затверд.							

Мета роботи: ознайомитися з ООП, множинним наслідуванням, міксинами в мові Python

Хід роботи:

Завдання на лабораторну роботу:

1. Створіть клас Alphabet. Його метод __init __ (), буде мати визначені два параметри: lang - мова і letters - список букв. Значення змінних lang і letters будуть визначенні за замовчуванням і міститимуться у вигляді статичних атрибутів для української мови. Клас матиме метод метод print_alphabet(), який виведе в консоль літери украйхнського алфавіту. Метод letters_num(), повертатиме кількість букв в алфавіті. Метод is_ua_lang() прийматиме довільний текст і визначатиме чи відноситься він до української мови (незалежно від регістру). Створіть клас EngAlphabet шляхом успадкування від класу Alphabet. Для його методу __init __(), всередині якого буде викликатися батьківський метод __init __(), в якості параметрів будуть передаватися позначення мови (наприклад, 'En') і рядок, що складається з усіх букв алфавіту. Додайте приватний статичний атрибут __en_letters_num, який буде зберігати кількість букв в алфавіті. Створіть метод is_en_letter(), який буде приймати строку в якості параметра і визначати, чи відноситься ця строка до англійського алфавіту. Перевизначити метод letters_num() - нехай в поточному класі класі він буде повертати значення властивості __en_letters_num. 6. Створіть статичний метод example(), який буде повертати приклад тексту англійською мовою.

   Тести до модуля:

   - Створіть об'єкт класу EngAlphabet
   - Надрукуйте літери алфавіту для цього об'єкту
   - Виведіть кількість букв в алфавіті
   - Перевірте, чи відноситься буква J до англійського алфавіту.
   - Перевірте, чи відноситься буква Щ до українського алфавіту
   - Виведіть приклад тексту англійською мовою

Лістинг програми:

```
""" Lab 9. Python. Andrii Babushko. Repository: https://github.com/AndriiBabushko/Python
"""
from __future__ import annotations
from typing import TextIO
from matplotlib import pyplot as plt
import csv
import os.path
import string

# task 1

class Alphabet:
```

```
    def __init__(self, lang: str = 'UA', letters=None):
        if letters is None:
            letters = ['А', 'Б', 'В', 'Г', 'Ґ', 'Д', 'Е', 'Є', 'Ж', 'З', 'И', 'І', 'Ї',
'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', 'Р', 'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш',
                       'Щ', 'Ь', 'Ю', 'Я']
        self.language: str = lang
        self.letters: list = letters

    def print_alphabet(self) -> None:
        alphabet_letters = ', '.join(self.letters)
        print(f'{self.language} alphabet letters: {alphabet_letters}')

    def letters_num(self) -> int:
        return len(self.letters)

    def is_ua_lang(self, text: str) -> bool:
        text = text.upper()
        text_list: list = list(text)

        if text_list[0] in self.letters:
            return True

        return False


class EngAlphabet(Alphabet):
    __en_letters_num: int = 26

    def __init__(self, lang: str, letters: list):
        super().__init__(lang, letters)

    def is_en_letter(self, eng_text: str) -> bool:
        eng_text = eng_text.upper()
        eng_text_list: list = list(eng_text)

        if eng_text_list[0] in self.letters:
            return True

        return False

    def letters_num(self) -> int:
        return self.__en_letters_num

    @staticmethod
    def example() -> str:
        return "Example:\nEnglish is the Language of International Communication.\nAlthough
English is not the most spoken language in the world,\nit is the official language in" \
               "53 countries and is spoken as a\nfirst language by around 400 million
people worldwide.\nBut that's not all, it is also the most common second language in the
world."


print('TASK 1!!!!')
print('ENG!')
task_1_eng_alphabet: EngAlphabet = EngAlphabet('ENG', list(string.ascii_uppercase))
task_1_eng_alphabet.print_alphabet()
print(f'Count letters in {task_1_eng_alphabet.language} alphabet:
{task_1_eng_alphabet.letters_num()}')
print(task_1_eng_alphabet.example())
if task_1_eng_alphabet.is_en_letter('J'):
    print('Letter \'J\' is in ENG alphabet!')
else:
    print('Letter \'J\' isn\'t in ENG alphabet!')

print('\nUA!')
task_1_alphabet: Alphabet = Alphabet()
task_1_alphabet.print_alphabet()
print(f'Count letters in {task_1_alphabet.language} alphabet:
{task_1_alphabet.letters_num()}')
```

```python
if task_1_alphabet.is_ua_lang('Щ'):
    print('Letter \'Щ\' is in UA alphabet!')
else:
    print('Letter \'Щ\' isn\'t in UA alphabet!')
```

Результат програми:

```
TASK 1!!!


ENG!
ENG alphabet letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
Count letters in ENG alphabet: 26
Example:
English is the Language of International Communication.
Although English is not the most spoken language in the world,
it is the official language in53 countries and is spoken as a
first language by around 400 million people worldwide.
But that's not all, it is also the most common second language in the world.
Letter 'J' is in ENG alphabet!


UA!
UA alphabet letters: А, Б, В, Г, Ґ, Д, Е, Є, Ж, З, И, І, Ї, Й, К, Л, М, Н, О, П, Р, С, Т, У, Ф, Х, Ц, Ч, Ш, Щ, Ь, Ю, Я
Count letters in UA alphabet: 33
Letter 'Щ' is in UA alphabet!
```

2. Створіть клас Human. Визначте для нього два статичних атрибути: default_name і default_age. Його метод __init __(), який крім self приймає ще два публічних параметри(name і age) і два приватних (money і house). Параметр money визначатиме кількість грошей, а house – посилання на об’єкт класу House. Метод info(), має виводити поля name, age, house і money. Реалізуйте довідковий статичний метод default_info(), який буде виводити статичні поля default_name і default_age. Реалізуйте приватний метод make_deal(), який буде відповідати за технічну реалізацію покупки будинку: зменшувати кількість грошей на рахунку і привласнювати посилання на тільки що куплений будинок. В якості аргументів даний метод приймає об'єкт будинку та його ціну. Реалізуйте метод earn_money(), що збільшує значення поля money. Реалізуйте метод buy_house(), який буде перевіряти, що у людини достатньо грошей для покупки, і здійснювати операцію. Якщо грошей занадто мало - потрібно вивести попередження в консоль. Параметри методу: посилання на будинок і розмір знижки (за замовчуванням 10%). Створіть клас House. Його метод __init __() містить два динамічних параметри: _area і _price, що мають значення за замовчуваннями. Створіть метод final_price(), який приймає як параметр розмір знижки і повертає ціну з урахуванням даної знижки. Створіть клас SmallHouse, успадкувавши його функціонал від класу House. Всередині класу SmallHouse перевизначите метод __init __() так, щоб він створював об'єкт з площею 40м2

   Тести до модуля:

   - Викличте довідковий метод default_info() для класу Human

- Створіть об'єкт класу Human
- Виведіть довідкову інформацію про створений об'єкт (викличте метод info ()).
- Створіть об'єкт класу SmallHouse
- Спробуйте купити створений будинок, переконайтеся в отриманні попередження.
- Виправте фінансове становище об'єкта - викличте метод earn_money()
- Знову спробуйте купити будинок
- Подивіться, як змінився стан об'єкта класу Human.

Лістинг програми:

```
# task 2

class House:
    def __init__(self, area: int = 50, price: float = 250000) -> None:
        self._area: int = area
        self._price: float = price

    def final_price(self, discount=0) -> float:
        return self._price - (self._price * discount / 100)

    def __str__(self) -> str:
        return f'Area: {self._area}; Price: {self._price};'

    def get_price(self) -> float:
        return self._price


class SmallHouse(House):
    def __init__(self) -> None:
        self.area = 40
        super().__init__(self.area)


class Human:
    default_name: str = 'Andrey'
    default_age: int = 18

    def __init__(self, name: str, age: int, money: float = 0, house: House = None) -> None:
        self.name: str = name
        self.age: int = age
        self.__money: float = money
        self.__house: House = house

    def info(self) -> str:
        return f'Info:\nName: {self.name}; Age: {self.age}; Money: {self.__money}; House:
{self.__house}'

    @staticmethod
    def default_info() -> str:
        return f'Default info:\nDefault name: {Human.default_name}; Default age:
{Human.default_age};'

    def __make_deal(self, house: House, price: float = 0) -> tuple:
        self.__money -= price
        self.__house = house
        return self.__money, self.__house

    def earn_money(self, salary: float) -> float:
```

```
        print('Earning money...')
        self.__money += salary
        return self.__money

    def buy_house(self, house, price: float, discount: int = 10) -> tuple:
        print('Buying house...')
        price: float = price - (price * discount / 100)
        if price <= self.__money:
            print('House has been bought!')
            self.__house = house
            self.__money -= price
            return self.__money, self.__house
        else:
            print("House hasn't been bought!")
            return self.__money, self.__house


print('\nTASK 2!!!')

print(Human.default_info())
task_2_human: Human = Human('Andrii', 18, 550000.0)
print(task_2_human.info())

task_2_small_house: SmallHouse = SmallHouse()
task_2_human.buy_house(task_2_small_house, task_2_small_house.get_price())
print(task_2_human.info())

task_2_human.earn_money(300000.0)
print(task_2_human.info())

task_2_house: House = House(100, 800000)
task_2_human.buy_house(task_2_house, task_2_house.get_price())
task_2_house: House = House(75, 600000)
task_2_human.buy_house(task_2_house, task_2_house.get_price())
print(task_2_human.info())
```

Результат програми:

```
TASK 2!!!
Default info:
Default name: Andrey; Default age: 18;
Info:
Name: Andrii; Age: 18; Money: 550000.0; House: None
Buying house...
House has been bought!
Info:
Name: Andrii; Age: 18; Money: 325000.0; House: Area: 40; Price: 250000;
Earning money...
Info:
Name: Andrii; Age: 18; Money: 625000.0; House: Area: 40; Price: 250000;
Buying house...
House hasn't been bought!
Buying house...
House has been bought!
Info:
Name: Andrii; Age: 18; Money: 85000.0; House: Area: 75; Price: 600000;
```

3. Створіть клас Apple. Його статичний атрибут states, яке буде містити всі стадії дозрівання яблука («Відсутнє», «Цвітіння», «Зелене», «Червоне»). Метод __init __(), всередині якого будуть визначені два динамічних protected атрибути: _index (номер яблука) і _state (приймає перше значення зі словника states). Створіть метод grow(), який буде переводити яблуко на наступну стадію дозрівання Створіть метод is_ripe(), який буде перевіряти, що яблуко дозріло (досягло останньої стадії дозрівання). Створіть клас AppleTree. Визначте метод __init __(), який буде приймати як параметр кількість яблук і на його основі буде створювати список об'єктів класу Apple. Даний список буде зберігатися всередині динамічного атрибуту apples. Створіть метод grow_all(), який буде переводити всі об'єкти зі списку яблук на наступний етап дозрівання. Створіть метод all_are_ripe(), який буде повертати True, якщо все яблука зі списку стали стиглими. Створіть метод give_away_all(), який буде чистити список яблук після збору врожаю Створіть клас Gardener. Його метод __init __(), міститиме два динамічних атриути: name (ім’я садівника, публічний атрибут) і _tree (приймає об’єкт класу AppleTree). Створіть метод work(), який змушує садівника працювати, що дозволяє яблукам ставати більш стиглими. Створіть метод harvest(), який перевіряє, чи всі плоди дозріли. Якщо всі - садівник збирає урожай. Якщо і - метод друкує попередження. Створіть статичний метод apple_base(), який виведе в консоль довідку з кількості яблук і ступені їх стиглості.

Тести до модуля:

- Створіть декілька об’єктів класу Apple.
- Викличте довідку по всім наявним яблукам
- Створіть об'єкти класів AppleTree і Gardener
- Використовуючи об'єкт класу Gardener, попрацювати над яблучним деревом.
- Спробуйте зібрати урожай
- Якщо яблука ще не дозріли, продовжуйте доглядати за деревом
- Зберіть урожай.

Лістинг програми:

```
# task 3

class Apple:
    states: dict[int, str] = {0: 'Відсутнє', 1: 'Цвітіння', 2: 'Зелене', 3: 'Червоне'}

    def __init__(self, index: int, state: int = 0) -> None:
        self._index: int = index
        self._state: int = state

    def grow(self) -> None:
        if self._state < 3:
            self._state += 1

    def is_ripe(self) -> bool:
        if self._state == 3:
            return True
        return False

    def info(self) -> str:
        return f'Apple number: {self._index}; Apple state: {self._state}'


class AppleTree:
    def __init__(self, apples_count: int) -> None:
        self.apples: list[Apple] = [Apple(apple_number) for apple_number in range(0,
apples_count)]

    def grow_all(self) -> None:
        for apple in self.apples:
            apple.grow()

    def all_are_ripe(self) -> bool:
        for apple in self.apples:
            if not apple.is_ripe():
                return False
        return True

    def give_away_all(self) -> bool:
        if self.all_are_ripe():
            self.apples = []
            return True
        return False


class Gardener:
    def __init__(self, name: str, apple_tree: AppleTree) -> None:
        self.name: str = name
        self._tree: AppleTree = apple_tree
```

```
    def work(self) -> None:
        print(f'{self.name}\'s working in process...')
        self._tree.grow_all()

    def harvest(self) -> bool:
        print('Try to harvest crop...')
        if self._tree.all_are_ripe():
            print('The crop is harvested!')
            self._tree.give_away_all()
            return True
        else:
            print('The crop is not harvested!')
            return False

    @staticmethod
    def apple_base() -> None:
        print('Here is a guide to play this little class game:\n'
              'Each apple has 4 states: '
              '*) None\n'
              '*) Flowering\n'
              '*) Green\n'
              '*) Red\n'
              'In order for apples to grow, you need to make the gardener work!')


print('\nTASK 3!!!')
task_3_apple_1: Apple = Apple(1)
task_3_apple_1.info()
task_3_apple_2: Apple = Apple(2)
task_3_apple_2.info()
task_3_apple_3: Apple = Apple(3)
task_3_apple_3.info()

Gardener.apple_base()
task_3_apple_tree: AppleTree = AppleTree(10)
task_3_gardener: Gardener = Gardener('Andrii', task_3_apple_tree)
attempt: int = 1
while True:
    print(f'Attempt #{attempt}')
    if task_3_gardener.harvest():
        break
    else:
        task_3_gardener.work()
        attempt += 1
```

Результат програми:

```
TASK 3!!!
Here is a guide to play this little class game:
Each apple has 4 states: *) None
*) Flowering
*) Green
*) Red
In order for apples to grow, you need to make the gardener work!
Attempt #1
Try to harvest crop...
The crop is not harvested!
Andrii's working in process...
Attempt #2
Try to harvest crop...
The crop is not harvested!
Andrii's working in process...
Attempt #3
Try to harvest crop...
The crop is not harvested!
Andrii's working in process...
Attempt #4
Try to harvest crop...
The crop is harvested!
```

4. Створіть клас KmrCsv, який має два атрибути класу за замовчуванням: ref (посилання на CSV файл з оцінками) і num (номер КМР), та методи для встановлення і, відповідно, визначення посилання на файл з оцінками, встановлення номеру КМР, читання файлу з оцінками та виведення інформації про файл (номер КМР і кількість студентів, що її виконали).

   Створіть клас Statistic, що містить наступні методи:

   - avg_stat() визначає відсотки правильних відповідей на кожне питання серед усіх студентів і повертає результат у вигляді кортежу чисел;

   - метод marks_stat() визначає яку оцінку набрала відповідна кількість студентів і повертає результат у викляді словника формату {оцінка: кількість студентів};

   - метод marks_per_time() визначає який середній бал за хвилину набирав студент за під час виконання КМР і повертає результат у вигляді словника формату {id студента (це перша колонка csv файлу): середній бал за хвилину} ;

   - метод best_marks_per_time(), який приймає два аргументи bottom_margin і top_margin (нижня і верхня межа вибірки підсумкових балів за КМР), та

формує для цієї вибірки п’ять найкращих результатів середніх балів за хвилину у вигляді кортежу формату (id студента, підсумкова оцінка, середній бал за хвилину).

Створіть клас Plots, що містить наступні методи:

- set_cat() – встановлює каталог в який зберігатимуться отримані графіки;
- avg_plot() – приймає кортеж з відсотками правильних відповідей на кожне окреме питання, формує гістограму на його основі і зберігає отриманий графік;
- marks_plot() – приймає словник з оцінками і кількістю студентів, що їх набрали, формує на його основі гістрограму і зберігає її
- best_marks_plot() – формує для п’яти найкращих результатів середніх балів за хвилину гістрограму і зберігає її.

Створіть клас KmrWork, що успадковує класи CsvKmr, Statistic і Plots. В якості аргументів екземпляр класу приймає посилання на csv файл та номер КМР.

Клас KmrWork містить наступні статичні атрибути

- kmrs - в ньому зберігається словник формату {номер КМР: адреса відповідного csv файла}
- cat – каталог для збереження результатів роботи

Крім успадкованих, клас KmrWork містить наступні методи:

- compare_csv() – виводить на екран і зберігає в txt файл результат порівняння статистики двох КМР (кількість виконаних КМР, середній бал за КМР, середній час виконання КМР);
- compare_avg_plots() – виводить на екран і зберігає дві гістограми з відсотками правильних відповідей на кожні окремі питання.

Тести до модуля:

- Створіть об’єкти kmr1 і kmr2 класу KmrWork.
- Використайте для об’єкту kmr2 методи avg_plot() і marks_plot()
- Для класу KmrWork використайте методи compare_csv() і compare_avg_plots().

Лістинг програми:

```
# task 4

class KmrCsv:
    import os

    def __init__(self, ref: str, num: int):
        if self.os.path.isfile(ref):
            self.ref: str = ref
            self.number: int = num
```

```
        else:
            print('File does not exist!')
            self.ref = None
            self.number = None

    def _get_read_lines(self) -> list:
        import io
        try:
            if os.path.isfile(self.ref):
                with io.open(rf'{self.ref}', 'rt', encoding='utf-8') as krm_csv:
                    marks_csv = csv.reader(krm_csv)
                    return [line for line in marks_csv]
            else:
                raise Exception('Can\'t find file!')
        except Exception as exc:
            print('Caught this error: ' + repr(exc))

    def _get_students_id(self) -> list:
        students: list = self._get_read_lines()

        students_id: list = []
        for student in students:
            students_id.append(student[0])

        return students_id

    def _get_student_answers(self) -> list:
        student_answers: list = []
        marks: list = self._get_read_lines()

        for answers in marks:
            student_answers.append(answers[5:])

        return student_answers

    def _get_student_marks(self) -> list:
        kmr_marks: list = []
        marks: list = self._get_read_lines()

        for i in range(len(marks)):
            mark_string = marks[i][4].split(',')
            mark_number = float(mark_string[0]) + float(int(mark_string[1]) / 100)
            kmr_marks.append(mark_number)

        return kmr_marks

    def _get_student_count(self) -> int:
        return len(self._get_read_lines())

    def file_info(self):
        if not os.path.isdir(r'./task4'):
            os.mkdir(r'./task4')

        marks = self._get_read_lines()

        print(f'Kmr number: {self.number}; Count of student who pass KMR: {len(marks)}')


class Statistics(KmrCsv):
    def __init__(self, ref: str, num: int) -> None:
        super().__init__(ref, num)

    @staticmethod
    def __remove_dict_duplicates(dictionary: dict) -> dict:
        result: dict = {}

        for key, value in dictionary.items():
            if value not in result.values():
                result[key] = value
```

```
        return result

    def avg_stat(self) -> tuple:
        numbers_list: list = []
        student_answers: list = list(self._get_student_answers())

        answer_counter: int = 0
        while answer_counter < len(student_answers[0]):
            correct_answers: int = 0
            for answer in student_answers:
                if answer[answer_counter] == '0,50':
                    correct_answers += 1

            answer_counter += 1
            numbers_list.append(round((correct_answers / len(student_answers)) * 100))

        return tuple(numbers_list)

    def marks_stat(self) -> dict:
        certain_marks_dict: dict = {}

        kmr_marks: list = self._get_student_marks()
        student_counter: int = 0

        for mark in kmr_marks:
            for mark_check in kmr_marks:
                if mark_check == mark:
                    student_counter += 1
            certain_marks_dict[mark] = student_counter
            student_counter = 0

        certain_marks_dict: dict = self.__remove_dict_duplicates(certain_marks_dict)

        return certain_marks_dict

    def marks_per_time(self) -> dict:
        students_time: list = []
        marks: list = self._get_read_lines()
        students_marks: list = self._get_student_marks()

        for i in range(self._get_student_count()):
            time: str = marks[i][3].split(' ')
            if len(time) > 2:
                minutes = int(time[0])
                seconds = int(time[2]) + minutes * 60
            else:
                minutes = int(time[0])
                seconds = minutes * 60
            students_time.append(seconds)

        students_mark_per_min: dict = {}
        students_id: list = self._get_students_id()

        for i in range(0, self._get_student_count()):
            students_mark_per_min[students_id[i]] = (round((students_marks[i] /
students_time[i]) * 60, 2))

        return students_mark_per_min

    def best_marks_per_time(self, bottom_margin: float = 0, top_margin: float = 10) ->
dict:
        students_code: list = list(self.marks_per_time().keys())
        student_marks: list = self._get_student_marks()
        marks_per_time: list = list(self.marks_per_time().values())

        interval_students_code: list = []
        interval_student_marks: list = []
        interval_marks_per_time: list = []
```

```
        for i in range(0, len(marks_per_time)):
            if bottom_margin <= student_marks[i] <= top_margin:
                interval_students_code.append(students_code[i])
                interval_student_marks.append(student_marks[i])
                interval_marks_per_time.append(marks_per_time[i])

        top_5_results: dict = {}
        for i in range(0, 5):
            max_average_mark = max(interval_marks_per_time)
            index_of_student = interval_marks_per_time.index(max_average_mark)
            top_5_results[interval_students_code[index_of_student]] =
f'{interval_student_marks[index_of_student]}, {interval_marks_per_time[index_of_student]}'
            interval_marks_per_time.pop(index_of_student)
            # print(f'Top {i + 1}! {index_of_student + 1} student has
{top_5_results[i]}/min mark.')
        return top_5_results


class Plots(Statistics):
    def __init__(self, ref: str, num: int, work_dir: str) -> None:
        super().__init__(ref, num)
        self.work_dir_ref: str = work_dir

    def set_cat(self, dir_name: str) -> None:
        if not os.path.isdir(rf'./{dir_name}'):
            os.mkdir(self.work_dir_ref)

    def avg_plot(self) -> None:
        avg_stat: tuple = self.avg_stat()
        answers: tuple = tuple(answer for answer in range(1, len(avg_stat) + 1))
        plt.plot(answers, avg_stat)
        plt.ylabel('Average statistics')
        plt.xlabel('Answers')
        plt.savefig(rf'{self.work_dir_ref}/avg_plot.png')
        plt.clf()

    def marks_plot(self) -> None:
        marks_stat: dict = self.marks_stat()
        plt.plot(list(marks_stat.keys()), list(marks_stat.values()))
        plt.ylabel('Total count of marks')
        plt.xlabel('Marks')
        plt.savefig(rf'{self.work_dir_ref}/marks_plot.png')
        plt.clf()

    def best_marks_plot(self) -> None:
        best_marks_per_time: dict = self.best_marks_per_time(1, 9)
        marks: list = []
        avg_marks: list = []
        for best_mark in best_marks_per_time.values():
            marks.append(best_mark.split(', ')[0])
            avg_marks.append(best_mark.split(', ')[1])

        plt.plot(marks, avg_marks)
        plt.ylabel('Average marks per time')
        plt.xlabel('Marks')
        plt.savefig(rf'{self.work_dir_ref}/best_marks_plot.png')
        plt.clf()


class KmrWork(Plots):
    import io

    def __init__(self, ref: str, num: int, work_dir: str) -> None:
        super().__init__(ref, num, work_dir)

    @staticmethod
    def __get_seconds_from_str(file_read_lines: list[str]) -> int:
        time_sec: int = 0
        for line in file_read_lines:
```

```
            time: list[str] = line[3].split(' ')
            if len(time) > 2:
                minutes = int(time[0])
                seconds = int(time[2]) + minutes * 60
                time_sec += minutes * 60 + seconds
            else:
                minutes = int(time[0])
                time_sec += minutes * 60

        return time_sec

    @staticmethod
    def __get_max_kmr_time(file_read_lines: list[str]) -> int:
        time_list: list = []
        for line in file_read_lines:
            time: list[str] = line[3].split(' ')
            if len(time) > 2:
                minutes = int(time[0])
                seconds = int(time[2]) + minutes * 60
                time_list.append(minutes * 60 + seconds)
            else:
                minutes = int(time[0])
                time_list.append(minutes * 60)

        max_time_sec: int = max(time_list)
        return round(max_time_sec)

    @staticmethod
    def __check_kmr_print(file_variable: TextIO, first_kmr: KmrWork, second_kmr: KmrWork,
message: str, first_check, second_check):
        if first_check > second_check:
            output_line: str = f'{message} KMR #{first_kmr.number}({first_check}) > KMR
#{second_kmr.number}({second_check})!'
            file_variable.write(output_line + '\n')
            print(output_line)
        elif first_check < second_check:
            output_line: str = f'{message} KMR #{first_kmr.number}({first_check}) < KMR
#{second_kmr.number}({second_check})!'
            file_variable.write(output_line + '\n')
            print(output_line)
        else:
            output_line: str = f'{message} KMR #{first_kmr.number}({first_check}) == KMR
#{second_kmr.number}({second_check})!'
            file_variable.write(output_line + '\n')
            print(output_line)

    def compare_csv(self, kmr_work: KmrWork) -> None:
        first_kmr_work_count: int = self._get_student_count()
        second_kmr_work_count: int = kmr_work._get_student_count()

        first_student_marks: list[float] = self._get_student_marks()
        first_student_marks_sum: float = 0
        for mark in first_student_marks:
            first_student_marks_sum += mark
        first_students_avg_mark: float = round(first_student_marks_sum /
len(first_student_marks), 3)

        second_student_marks: list[float] = kmr_work._get_student_marks()
        second_student_marks_sum: float = 0
        for mark in second_student_marks:
            second_student_marks_sum += mark
        second_students_avg_mark: float = round(second_student_marks_sum /
len(second_student_marks), 3)

        first_read_lines: list = self._get_read_lines()
        first_time_sec: int = self.__get_seconds_from_str(first_read_lines)

        second_read_lines: list = kmr_work._get_read_lines()
        second_time_sec: int = self.__get_seconds_from_str(second_read_lines)
```

```
        first_avg_time: float = round(float(first_time_sec) / first_kmr_work_count / 60, 3)
        second_avg_time: float = round(float(second_time_sec) / second_kmr_work_count / 60,
3)

        if not os.path.isdir(rf'./{self.work_dir_ref}'):
            os.mkdir(rf'./{self.work_dir_ref}')

        with self.io.open(rf'./{self.work_dir_ref}/compare_csv_txt.txt', 'wt',
encoding='utf-8') as compare_csv_txt:
            KmrWork.__check_kmr_print(compare_csv_txt, self, kmr_work, 'The number of
completed', first_kmr_work_count, second_kmr_work_count)
            KmrWork.__check_kmr_print(compare_csv_txt, self, kmr_work, 'The average mark
of', first_students_avg_mark, second_students_avg_mark)
            KmrWork.__check_kmr_print(compare_csv_txt, self, kmr_work, 'The average time
of', first_avg_time, second_avg_time)

    def compare_avg_plots(self, kmr_work: KmrWork):
        first_avg_stat: tuple = self.avg_stat()
        second_avg_stat: tuple = kmr_work.avg_stat()
        answers: tuple = tuple(answer for answer in range(1, len(first_avg_stat) + 1))

        plt.plot(answers, first_avg_stat)
        plt.ylabel('Average statistics')
        plt.xlabel('Answers')
        plt.savefig(rf'{self.work_dir_ref}/first_avg_plot.png')
        plt.clf()

        plt.plot(answers, second_avg_stat)
        plt.ylabel('Average statistics')
        plt.xlabel('Answers')
        plt.savefig(rf'{self.work_dir_ref}/second_avg_plot.png')
        plt.clf()


print('\nTASK 4!!!')
task_4_kmr1: KmrWork = KmrWork(r'./task4/marks.lab6.csv', 6, 'test')
task_4_kmr2: KmrWork = KmrWork(r'./task4/marks2.lab11.csv', 11, 'test')
task_4_kmr2.avg_plot()
task_4_kmr2.marks_plot()
task_4_kmr1.compare_csv(task_4_kmr2)
task_4_kmr1.compare_avg_plots(task_4_kmr2)
```

Результат програми:

```
TASK 4!!!
D:\Політех\2_course_1_semester\Python\Lab9\lab9.py:519:
  plt.plot(answers, avg_stat)
The number of completed KMR #6(170) > KMR #11(112)!
The average mark of KMR #6(8.521) < KMR #11(9.288)!
The average time of KMR #6(27.462) < KMR #11(40.209)!
```

Висновок: під час виконання лабораторної роботи було отримано навички створення батьківських та дочірніх класів з використанням наслідування, використання приватних за захищених методів і атрибутів класу, створення статичних та звичайних методів класу, а також побудова графіків в залежності від вхідних даних.