

Загальна мета проекту до 1 та 2 лабораторних робіт

Метою даної лабораторної роботи було:

1. Освоїти базові принципи об'єктно-орієнтованого програмування (ООП): робота з класами, методами, полями, використання інкапсуляції, наслідування, поліморфізму (динамічного і статичного).
 2. Продемонструвати уміння розробляти об'єктну модель із кількома ієрархіями класів, реалізувати нетривіальні методи, а також забезпечити інтеграцію з графічним інтерфейсом.
 3. Виконати умови щодо кількості класів, полів, методів та поліморфізму, а також розширити програму додаванням графічного інтерфейсу користувача (GUI), який відповідає додатковим вимогам (4 вікна, 20+ елементів керування, контейнерні елементи, 10+ обробників подій).
-

Початкові умови та вимоги

До коду без GUI:

- **Мінімум 9 класів.**
- **Не менше 15 полів** у сукупності по всіх класах.
- **Принаймні 25 нетривіальних методів.**
- **2 ієрархії наслідування:**
 - Перша ієрархія має включати не менше 3 класів (наприклад, `Person` -> `Student` -> `Teacher` -> `Administrator`).
 - Друга ієрархія (наприклад, `Course` -> `OnlineCourse`, `OnsiteCourse`).
- **Поліморфізм:**
 - Динамічний поліморфізм: перевизначення методів у спадкоємцях.
 - Статичний поліморфізм: реалізовано через параметри за замовчуванням та використання узагальнених підходів (у Python немає класичної перевантаженості методів, але ми можемо імітувати її через `*args`, `**kwargs`).
- Інкапсуляція, принципи ОО-дизайну, відсутність дублювання коду.

Додаткові умови для GUI:

- **Не менше 4 вікон (форм):** головне вікно вибору ролі, вікно адміністратора, вікно викладача, вікно студента.
- Сукупно **20+ елементів керування** (кнопки, поля вводу, випадаючі списки, списки, тощо).
- **1 контейнерний елемент керування** (наприклад, `Listbox`, `Treeview` або `Combobox`).

- Не менше **10 обробників подій** (події натискання кнопок, вибору зі списку, тощо).
 - Розділення логіки (ООП-модель, логіка взаємодії) від коду GUI.
 - Документація та відповідність ОО-принципам.
-

Структура проекту

Структура файлів та каталогів приблизно така:

```
project/
├── models/
│   ├── person.py           # Базовий клас Person
│   ├── student.py         # Клас Student (наслідує Person)
│   ├── teacher.py         # Клас Teacher (наслідує Person)
│   ├── administrator.py   # Клас Administrator (наслідує Person)
│   ├── course.py          # Клас Course (базовий)
│   ├── online_course.py   # Клас OnlineCourse (наслідує Course)
│   ├── onsite_course.py    # Клас OnsiteCourse (наслідує Course)
│   ├── assignment.py      # Клас Assignment
│   ├── schedule.py        # Клас Schedule для роботи з розкладом
│   └── ... (інші залежні класи, якщо є)
├── system_interface.py     # Інтерфейс взаємодії без GUI (консольний)
├── gui.py                  # Графічний інтерфейс користувача на основі Tkinter
└── main.py                 # Точка входу: демонстративний та інтерактивний режими
```

Короткий опис основних класів та їх ролі

- **Person (models/person.py):**
Базовий клас, що представляє людину з базовими полями (ім'я, прізвище, email).
- **Student, Teacher, Administrator (models/student.py, teacher.py, administrator.py):**
Наслідують від Person.
 - Student: має студентський ID, перелік курсів, методи для зарахування на курс, перегляду та здачі завдань, перегляду оцінок.
 - Teacher: викладач, який має ID викладача, може створювати завдання, виставляти оцінки.
 - Administrator: відповідає за створення курсів, реєстрацію користувачів (студентів, викладачів), призначення викладача на курс.
- **Course, OnlineCourse, OnsiteCourse (models/course.py, online_course.py, onsite_course.py):**

Ієрархія курсів. **Course** – базовий клас. **OnlineCourse** та **OnsiteCourse** наслідують його та додають специфічні поля (платформа для онлайн або аудиторія для очного курсу).

- **Assignment (models/assignment.py):**
Завдання із полями: ID, назва, дедлайн, максимальна оцінка, списки submissions та grades.
- **Schedule (models/schedule.py):**
Клас для роботи з розкладом, дозволяє додавати події, переглядати події на сьогодні.
- **SystemInterface (system_interface.py):**
Логіка взаємодії в консольному режимі: реєстрація користувачів, створення курсів, призначення викладачів, перегляд даних тощо.
- **GUI (gui.py):**
Реалізація графічного інтерфейсу користувача за допомогою Tkinter. Включає вікно вибору ролі, меню адміністратора, меню викладача, меню студента з відповідними кнопками, полями вводу, списками. Обробляє події натиснення кнопок, вибору курсів, завдань і т.д.
- **main.py:**
Точка входу до програми. Пропонує вибір між демонстративним режимом (автоматичне виконання сценарію) та інтерактивним (запуск **SystemInterface** або запуск GUI, якщо це передбачено).
Демонстративний режим (**demo()**) створює прикладові об'єкти та демонструє логіку (створення завдання, запис студента, виставлення оцінок, перегляд GPA та розкладу).

Виконання вимог ООП

- **9+ класів:**
Класи **Person**, **Student**, **Teacher**, **Administrator**, **Course**, **OnlineCourse**, **OnsiteCourse**, **Assignment**, **Schedule** - вже більше 9 класів.
- **15+ полів у сумі:**
Сумарно в усіх класах є значно більше 15 полів (ім'я, прізвище, email у **Person**; ID студента, списки курсів і оцінок у **Student**; ID викладача, список курсів у **Teacher**; поля курсу (ID, title, description), поля завдання (title, due_date, max_grade), та інші).

- **25+ нетривіальних методів:**
Методи типу `enroll_course`, `submit_assignment`, `calculate_gpa`, `assign_grade`, `create_assignment`, `get_today_events`, `add_student`, `assign_teacher_to_course` виконують логічні операції з даними, а не лише гетери/сетери.
 - **2 ієрархії наслідування:**
 - `Person` -> `Student`, `Teacher`, `Administrator`
 - `Course` -> `OnlineCourse`, `OnsiteCourse`
В першій ієрархії більше 3 класів (`Person`, `Student`, `Teacher`, `Administrator`).
 - **Поліморфізм:**
 - Динамічний поліморфізм: наприклад, методи для відображення інформації про курс можуть бути перевизначені у `OnlineCourse` та `OnsiteCourse`.
 - Статичний поліморфізм: імітується через використання аргументів за замовчуванням, `*args`, `**kwargs` у деяких методах.
Також поліморфізм проявляється при викликах методів для різних об'єктів класів-нащадків `Person` чи `Course`.
 - **Інкапсуляція:**
Використання приватних та захищених атрибутів, надання методів для безпечного доступу до даних.
 - **Відсутність дублювання коду:**
Спільний функціонал винесено у базові класи. Наприклад, `Person` надає базові поля та методи для `Student` і `Teacher`.
-

Виконання вимог GUI

- **Не менше 4 вікон/форм:**
 1. Головне вікно (вибір ролі: адміністратор, викладач, студент)
 2. Вікно меню адміністратора
 3. Вікно меню викладача
 4. Вікно меню студента
- **20+ елементів керування:**
Кнопки, поля вводу (`Entry`), списки (`Listbox`), випадаючі списки (`Combobox`), метки (`Label`) - у сукупності значно більше 20.

- **1 контейнерний елемент керування:**
Використано `Listbox` та `Combobox` - це контейнерні/спискові елементи.
 - **10+ обробників подій:**
Є окремі методи для обробки натискання кнопок, обрання елементів зі списків, переходу між вікнами. Наприклад, `submit_register_teacher`, `submit_create_course`, `submit_assignment_submission` тощо - кожен викликається при певній дії користувача.
 - **Розділення логіки та GUI:**
Логіка розподілена по моделях та `system_interface.py` (консольна логіка), а `gui.py` опікується виключно відображенням та отриманням даних від користувача. Моделі (`Student`, `Teacher`, `Course` і т.д.) не залежать від коду інтерфейсу.
-

Демонстративний режим

Виклик `demo()` у `main.py` демонструє:

- Створення адміністратора, викладача, студента, курсів.
- Призначення викладача на курси.
- Запис студента на курси.
- Створення завдання викладачем.
- Здача завдання студентом.
- Виставлення оцінки викладачем.
- Обчислення середнього балу (GPA) студента.
- Роботу з розкладом (додавання подій та перегляд подій на сьогодні).

Це дозволяє переконатися у коректній роботі внутрішньої логіки без GUI.

Інтерактивний режим

Якщо користувач обирає інтерактивний режим, викликається `main()` та `system.run()`.

В `system_interface.py` користувачі можуть взаємодіяти з системою через консоль, реєструвати викладачів, студентів, створювати курси, призначати викладачів, переглядати курси тощо.

Якщо використовується `gui.py`, можна запустити графічний інтерфейс, де користувач натискає кнопки, вводить дані у поля, обирає зі списків курсів, завдань і так далі.

Відповідність вимогам

- Виконано ООП-вимоги: кількість класів, полів, методів, наявність двох ієрархій наслідування, застосування поліморфізму.
- Реалізовано GUI з 4 вікнами, 20+ елементами керування, контейнерними елементами, 10+ обробниками подій.
- Логіка відокремлена від інтерфейсу.
- Продемонстровано роботу системи як у демонстративному, так і в інтерактивному режимі.
- Код задокументований, доступні приклади використання (функція `demo()`).

Висновок

Даний проект повністю відповідає поставленим вимогам:

1. Реалізовано ООП-принципи (наслідування, поліморфізм, інкапсуляція).
2. Досягнуто необхідну кількість класів, полів, нетривіальних методів.
3. Створено дві ієрархії наслідування, включно з однією з трьома і більше класами.
4. Додано графічний інтерфейс з 4 вікнами, 20+ елементами керування, контейнерними елементами та 10+ обробниками подій.
5. Забезпечено розділення логіки (моделі, логіка системи) від графічного інтерфейсу, а також можливість демонстративного та інтерактивного режимів роботи.

https://github.com/khaenare/university/tree/main/semester_3/OOP/lab1