Lista zadań nr 2

Zadanie 1 Napisz program symulujący prosty notatnik. Program powinien składać się z modułu zawierającego klasy `Note` i `Notebook`, a także klasę `Menu`, która obsługuje interakcję z użytkownikiem.

Klasa `Note` reprezentuje pojedynczą notatkę i powinna posiadać następujące atrybuty:

- `text` - tekst notatki przekazany jako argument konstruktora,
- `tag` - etykieta (tag) przypisana do notatki,
- `date` - data utworzenia notatki (ustawiana automatycznie),
- `ID` - unikalny numer notatki w ramach notatnika (generowany automatycznie, np. poprzez atrybut klasy zliczający utworzone instancje).

Klasa powinna również zawierać metodę `match(self, query)`, która sprawdza, czy tekst notatki lub jej etykieta zawiera podane słowo kluczowe `query`. Metoda zwraca wartość logiczną `True` lub `False`.

Klasa `Notebook` reprezentuje zbiór notatek i powinna zawierać atrybut `notes`, czyli listę przechowującą obiekty klasy `Note`. Klasa ta powinna implementować następujące metody:

- `new_note(text, tag)` - dodaje nową notatkę do listy,
- `modify_text(ID, new_text)` - zmienia treść notatki o podanym `ID`,
- `modify_tag(ID, new_tag)` - zmienia etykietę notatki o podanym `ID`,
- `search(query)` - zwraca listę notatek zawierających dany ciąg znaków w treści lub etykiecie. Jeśli brak wyników, zwraca pustą listę.

Klasa `Menu` umożliwia interakcję użytkownika z notatnikiem. Powinna ona zawierać atrybut `notebook`, inicjalizowany nowym obiektem klasy `Notebook`, oraz atrybut `options` - słownik mapujący klawisze menu na odpowiednie metody:

```
{
    "1": self.show_notes,
    "2": self.search_notes,
    "3": self.add_note,
    "4": self.modify_note,
    "5": self.quit
}
```

Dodatkowo klasa powinna implementować następujące metody:

- `show_menu()` - wyświetla dostępne opcje menu,
- `run()` - obsługuje wybór użytkownika, odczytując odpowiednią metodę ze słownika `options`,
- `show_notes()` - wyświetla wszystkie notatki,
- `search_notes()` - prosi użytkownika o wpisanie szukanego tekstu i wyświetla znalezione notatki za pomocą metody `show_notes()`,
- `add_note()` - umożliwia dodanie nowej notatki przez użytkownika,
- `modify_note()` - pozwala użytkownikowi zmienić treść lub etykietę istniejącej notatki, wyszukując ją po `ID`,
- `quit()` - kończy działanie programu.

Uwaga: Metoda `search_notes()` powinna wykorzystywać metodę `show_notes()` do wyświetlania wyników wyszukiwania. Można to zrealizować np. poprzez przekazanie listy znalezionych notatek jako argumentu do metody `show_notes()`.

Zadanie 2 Napisz program, który tworzy obiekt klasy `Person`. Klasa ta powinna posiadać atrybuty: `name`, `surname` oraz `age`. Program ma umożliwiać uzupełnienie wartości tych atrybutów danymi wprowadzonymi przez użytkownika z klawiatury.

Wprowadzone dane należy zwalidować w następujący sposób:

- `age` - musi być liczbą całkowitą z zakresu od 0 do 130,
- `name` oraz `surname` - muszą zawierać co najmniej trzy znaki.

Do realizacji tej funkcjonalności wykorzystaj właściwości (*properties*). Klasa `Person` powinna również definiować metodę `__str__()`, która zwraca czytelny opis obiektu.

Następnie zaimplementuj dwie klasy: `Student` oraz `Employee`, dziedziczące po klasie `Person`.

- Klasa `Student` powinna posiadać dodatkowe atrybuty:
  - `major` - kierunek studiów,
  - `grades` - słownik, w którym kluczami są nazwy przedmiotów, a wartościami uzyskane oceny.
- Klasa `Employee` powinna posiadać dodatkowe atrybuty:
  - `position` - stanowisko pracy,

- `skills` - lista kluczowych umiejętności pracownika.

Zaimplementuj odpowiednie metody umożliwiające uzupełnianie oraz wyświetlanie wartości atrybutów w obu klasach potomnych.

Zadanie 3 Zaprojektuj klasę `Rectangle` z dwoma atrybutami: `length` i `height`, które reprezentują długości boków prostokąta. Klasa powinna posiadać następujące metody:

- `__init__()` - konstruktor klasy inicjalizujący atrybuty instancji,
- `area()` - zwraca pole prostokąta,
- `__str__()` - zwraca czytelną reprezentację obiektu w postaci tekstowej,
- `__repr__()` - zwraca jednoznaczną reprezentację obiektu, zgodną z ogólnie przyjętymi zasadami, napisaną tak, aby nie wymagała nadpisywania w klasach pochodnych.

Zdefiniuj klasę `Cuboid`, dziedziczącą po klasie `Rectangle`, która powinna posiadać dodatkowy atrybut `width` oraz następujące metody:

- `__init__()` - konstruktor klasy inicjalizujący wszystkie atrybuty, wywołujący konstruktor klasy bazowej,
- `area()` - zwraca pole powierzchni całkowitej prostopadłościanu (w tym celu wykorzystaj odpowiednią metodę klasy bazowej),
- `volume()` - zwraca objętość prostopadłościanu (również z wykorzystaniem metod klasy bazowej),
- `__str__()` - zwraca czytelną reprezentację obiektu w postaci tekstowej.

Inicjalizacja atrybutów instancji powinna odbywać się poprzez wartości przekazywane jako parametry konstruktora.
Uwaga: Metoda `__repr__()` w klasie `Rectangle` powinna być zaimplementowana zgodnie z ogólnie przyjętymi zasadami jej definiowania, tak aby nie było konieczności jej nadpisywania w klasach pochodnych.

Napisz program, który odczytuje dane z pliku tekstowego (*dane.txt*). Dane powinny być wczytywane aż do końca pliku. W kolejnych wierszach znajdują się:

- liczba 1 lub 2 (1 - prostokąt, 2 - prostopadłościan),
- następnie, oddzielone spacjami:
  - dla prostokąta: długość i wysokość,
  - dla prostopadłościanu: długość, wysokość i szerokość.

Program powinien wypisywać na ekranie typ figury, jej charakterystyczne parametry, pole powierzchni, a w przypadku prostopadłościanu również jego objętość.

Zaimplementuj obsługę wyjątków:

- Zdefiniuj własną klasę wyjątku `InvalidData`, dziedziczącą po `Exception`. Wykorzystaj ją do obsługi sytuacji wyjątkowych, takich jak ujemne lub zerowe wartości długości boków bądź krawędzi.
- Obsłuż błędy związane z operacjami wejścia/wyjścia (`IOError`).
- Obsłuż błędy wynikające z niepoprawnego formatu danych (np. błędny typ wartości w pliku).