Lista zadań nr 2

Zadanie 1 Napisz program, który będzie symulował bardzo prosty notatnik. Utwórz plik modułu, który zawiera klasy: `Note` i `Notebook`. Konstruktor pierwszej klasy powinien tworzyć i inicjalizować atrybuty `text` i `tag` (dane przekazane przez parametry konstruktora) atrybut `date` (automatyczna data tworzenia notatki) oraz atrybut `ID` (automatyczny numer notatki w danym notatniku - inicjalizowany przez odpowiedni atrybut klasy zliczający liczbę już utworzonych instancji klasy). Klasa `Note` powinna posiadać także metodę `match`, która zwraca `True` lub `False` jeżeli `text` lub `tag` zawiera przekazany do tej metody ciąg tekstowy. Konstruktor klasy `Notebook` powinien tworzyć i inicjalizować pustą listą - atrybut `notes`. Klasa `Notebook` powinna posiadać następujące metody: `new_note()` - pozwala na dodanie obiektu klasy `Note` do notatnika (listy `notes`); `modify_text()` - pozwalająca na zmianę tekstu notatki o podanym `ID`; `modify_tag()` - pozwalająca na zmianę tekstu etykiety notatki o podanym `ID`; `search()` - zwraca listę notatek zawierających szukaną frazę (w tekście lub etykiecie notatki). Główny program powinien importować klasy, które są przedstawione wyżej i definiować nową klasę `Menu`. Konstruktor klasy `Menu` powinien tworzyć następujące atrybuty: `notebook` - inicjalizowany obiektem klasy `Notebook`, `options` - inicjalizowany słownikiem: `{"1": self.show_notes, "2": self.search_notes, "3": self.add_note, "4": self.modify_note, "5": self.quit}`. We wspominanej klasie zdefiniuj metody: `show_menu()` - wyświetlająca menu notatnika; `run()` - zapewniająca pobranie odpowiedniego klucza i odczytanie odwadniającej mu wartości słownika `options`; metody odpowiadające wartościom słownika `options` tzn. `show_notes()`, `search_notes()` itd. (łatwo wywnioskować jak mają działać te metody). **Uwaga:** Metoda `search_notes()` powinna wyświetlać znalezioną listę notatek (zawierających szukaną frazę) za pomocą wywołania metody `show_notes()` (jak?).

Zadanie 2 Napisz program, który tworzy obiekt klasy `Person`. Klasa posiada atrybuty: `name`, `surname`, `age`. Program pozwala na uzupełnienie wartości atrybutów danymi podanymi z klawiatury. Wiek musi być liczbą całkowitą w zakresie od 0 do 130, a imię i nazwisko muszą posiadać minimum 3 znaki - wykorzystaj w tym celu właściwości. Klasa `Person` powinna definiować metodę `__str__()`. Następnie, zaimplementuj dwie klasy `Student` i `Employee`, oparte na klasie `Person`. W klasie `Student` dodaj atrybuty `field_of_study` i `student_book` - słownik, którego klucze to nazwy przedmiotów, a wartości to oceny. W klasie `Employee` dodaj atrybut `job_title` i `skills` - lista kluczowych umiejętności. Zaimplementuj odpowiednie metody pozwalające na uzupełnianie i wyświetlanie wartości atrybutów w obu klasach potomnych.

Zadanie 3 Zdefiniuj klasę `Rectangle` z dwoma atrybutami: `length` i `height` - długości boków. Klasa powinna posiada następujące metody:

- `__init__()`;
- `area()` - zwraca pole;
- `__str__()`;
- `__repr__()`.

Zdefiniuj klasę `Cuboid` dziedziczącą po klasie `Rectangle` i mającą dodatkowy atrybut `width` oraz metody:

- `__init__()` - wywołuje konstruktor klasy bazowej;
- `area()` - zwraca pole powierzchni prostopadłościanu (wykorzystaj odpowiednią metodę klasy bazowej);
- `volume()` - metoda ma zwracać objętość prostopadłościanu (wykorzystaj odpowiednią metodę klasy bazowej);
- `__str__()`;

Inicjalizacja atrybutów instancji klas powinna odbywać się poprzez wartości jej parametrów. **Uwaga:** Metoda `__repr__()` w klasie `Rectangle` powinna być napisana zgodnie z ogólnie przyjętymi zasadami jej tworzenie oraz w taki sposób aby nie było konieczności jej nadpisywania (i modyfikowania) w klasie pochodnej.

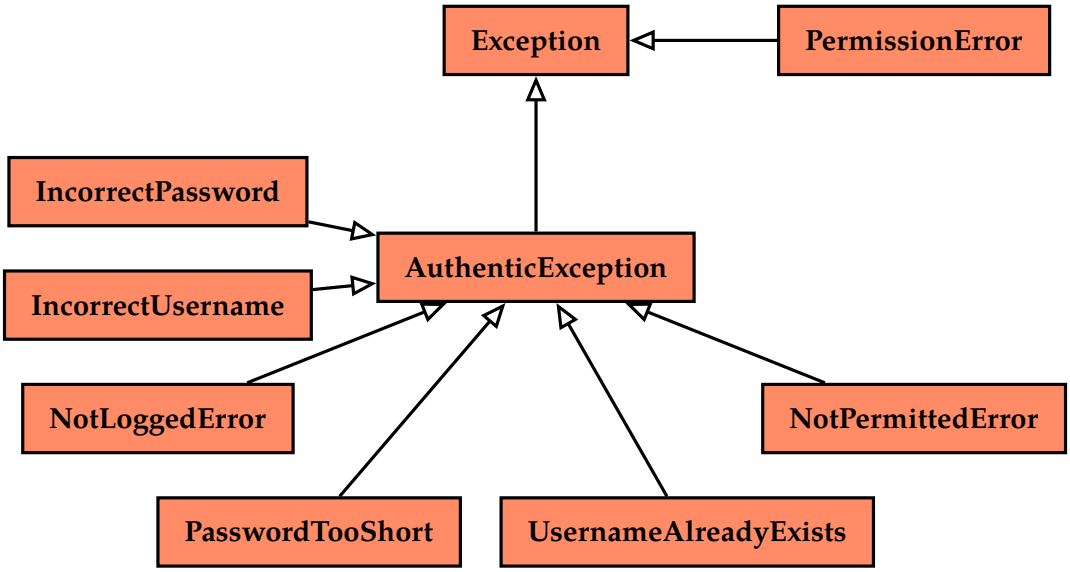
Napisz program, w której wczytasz dane z pliku tekstowego - dane czytane do końca pliku (*dane.txt*). W kolejnych wierszach pliku znajdują się: liczba 1 lub 2 (1-prostokąt, 2-prostopadłościan), a następnie oddzielone spacjami, w przypadku prostokąta długość i wysokość zaś w przypadku prostopadłościanu długość, wysokość i szerokość. Wypisz na ekranie monitora typ figury, parametry ją charakteryzujące i pole powierzchni, a dla prostopadłościanu także objętość. Zastosuj obsługę wyjątków. Zdefiniuj własną klasę `InvalidData` dziedzicząca po `Exception`. Wykorzystaj tę klasę do obsługi sytuacji wyjątkowych:
ujemne lub zerowe długości boków/krawędzi. Obsłuż błędy `IO` i błąd złego typu danych.

Zadanie 4 Napisz program, który symuluje prosty system uwierzytelniania i autoryzacji. W module `authorization_system` utwórz klasy:

- `User` - która przechowuje nazwę użytkownika i zaszyfrowane hasło. Klasa powinna zawierać metody:

- ◇ `__init__()` - tworzy i incjializuje atrybutu: `username` - nazwa użytkownika podana podczas konkretyzacji obiektu; `password` - hasło (złożone z nazwy użytkownika i hasła) podane podczas konkretyzacji obiektu, ale zaszyfrowane przez `_encrypt_password()`; `is_logged` - o początkowej wartości `False`;
- ◇ `_encrypt_password()` - metoda zmieniająca hasło (podane jako argument metody) i nazwę użytkownika na zaszyfrowaną wersję, którą zwraca (wykorzytaj moduł `hashlib` i np. funkcję skrótu `SHA-256`);
- ◇ `check_password()` - metoda sprawdzająca hasło (podane jako argument metody) z hasłem przechowywanym w atrybucie i zwracająca `True` lub `False`.

- prostą hierarichę klas wyjątków o pustych ciałach (zob poniższy diagram UML)



- klasę `Authenticator` - klasa kontrolująca użytkowników. Klasa powinna zawierać metody:
  - ◇ `__init__()` - tworzy i incjializuje pustym słownikiem atrybut `users`;
  - ◇ metoda `add_user`, która umożliwia dodanie użytkownika (o podanej nazwie i haśle) do słownika `user` pod warunkiem, że w słowniku nie ma takiego użytkownika (w przeciwnym wypadku następuje podsinienie wyjątku `UsernameAlreadyExists`) oraz jego hasło ma więcej niż 7 znaków (w przeciwnym wypadku następuje podsinienie wyjątku `PasswordTooShort`);
  - ◇ metoda `login`, która służy do logowania (gdy użytkownik jest zalogowany jago atrybut `is_logged` przyjmuje wartość `True`), metoda podnosi wyjątek `IncorrectUsername`, gdy taki użytkownik nie ma konta (nie ma go w słowniku `users`) lub wyjątek `IncorrectPassword`, gdy hasło użytkownika jest niepoprawne - gdy logowanie się powiedzie metoda powinna też zwrócić wartość `True`;

- ◇ metoda `is_logged_in` - zwraca odpowiednio `True` lub `False` gdy dany użytkownik jest lub nie jest zalogowany;

- klasę `Authorizor`, która mapuje uprawnienia dla użytkowników. Klasa powinna zawierać metody:

  - ◇ `__init__()` tworzy i incjializuje pustym słownikiem atrybut `permissions` oraz atrybut `authenticator` inicjalizowany parametrem konstruktora;
  - ◇ metoda `add_permission`, która pozwala dodać do słownika nowe uprawnienie jako klucz z wartością będącą pustym zbiorem, jeżeli uprawnienie już istnieje metoda podnosi wyjątek `PermissionError`
  - ◇ metoda `permit_user`, która umożliwia przypisanie danemu użytkownikowi podanemu jako argument metody odpowiedniego uprawnienia (drugi argument metody). Metoda powinna podnosić w odpowiednich miejscach wyjątki `PermissionError` oraz `IncorrectUsername`.
  - ◇ metoda `check_permission`, która pozwala sprawdzić czy podany użytkownik posiada wskazane uprawnianie. Metoda powinna podnosić odpowiednie wyjątki: `NotLoggedError` - gdy użytkownik nie jest zalogowany, `PermissionError` - gdy nie ma takiego uprawnienia, `NotPermittedError` - gdy użytkownik nie ma podanego uprawnienia.

Moduł zakończ stworzeniem instancji klasy `Authenticator` oraz `Authorizor` (argumentem konstruktora tej drugiej instancji jest oczywiście pierwszy obiekt).

W głównym programie utwórz odpowiednie instancje obiektów reprezentacyjnych użytkowników i nadaj im uprawnienia (np. testowania i/lub zmieniania programów). Stwórz klasę `Editor`, która zawiera podstawowy interfejs menu pozwalający niektórym użytkownikom zmienić lub testować program. Wspomniana klasa powinna zawierać metody:

- ◇ `__init__()` - tworzy dwa atrybutu: `username` wartości `None` oraz `options` o wartosści self.options = {"a": self.login, "b": self.test, "c": self.change, "d": self.quit}.
- ◇ `login()` - metoda pobierająca od użytkownika nazwę i hasło oraz wywołująca odpowiednią metodę `login()` stworzonej instancji klasy `Authenticator` wraz z obsługą wyjątków;
- ◇ `is_permitted()` - metoda sprawdzająca czy użytkownik jest zalogowany i ma odpowiednie uprawnienia (wywołuje metodę `check_permission` i obsługuje odpowiednie wyjątki);
- ◇ `test()` - metoda imitująca testowanie hipotetycznego programu (korzysta z metody `is_permitted()`);

- ◇ `change()` - metoda imitująca zmienianie hipotetycznego programu (korzysta z metody `is_permitted()`);
- ◇ `quit()` - metoda kończąca działanie głównego programu;
- ◇ `run()`, która zapewnia pobranie od użytkownika odpowiedniego klucza i odczytanie (wraz z wywołaniem) odwadniającej mu wartości słownika `options`.

Główny program powinien tworzyć instancje klasy `Editor` i wywoływać metodę `run()`.