Lista 3 :

Klasy, obiekty, dziedziczenie, przesłanianie, klasy abstrakcyjne, serializacja – Python

Zad. 1

Przygotuj rozbudowaną hierarchię dziedziczenia, np.:

klasa_Osoba; klasa_Student; klasa_parcownik_PWr; klasa_Pracownik_Administracyjny; klasa_Pracownik_Naukowy (lub inną, ale również rozbudowaną).

Określ cechy i zachowania dla obiektów tych klas.

Wszystkie klasy poza 'liści drzewa dziedziczenia', oznacz jako abstrakcyjne – aby zdefiniować klasy abstrakcyjne, skorzystaj z modułu *abc*.

Zachowaj zasady hermetyzacji – pracuj wyłącznie na polach prywatnych (notacja zmiennej prywatnej w Pythonie: *__zmiennaPrywatna*).

W korzeniu hierarchii dziedziczenia zdefiniuj co najmniej jedną metodę abstrakcyjną – np. wyświetlającą stan obiektu (wykorzystaj do tego dekorator: *@abstractmethod* z modułu *abc*).

Przesłoń w odpowiedni sposób powyższą metodę w klasach, dla których stworzysz konkretne obiekty.

Zainicjuj poprawne przekazanie informacji z konstruktorów (*__init__(self, ...)*) klas podrzędnych do konstruktorów klas nadrzędnych – wykorzystaj wywołanie *super()*.

Zad. 2

Dla tak zdefiniowanej hierarchii dziedziczenia zaimplementuj całą funkcjonalność z listy nr 2, rozszerzając ją również na pracowników PWr.

Pracuj na kolekcji danych określonej jako *lista obiektów*.

Zad. 3

Wykonaj serializację (zapis stanu obiektów do pliku binanrnego) zaimplementowanej kolekcji obiektów, przy wykorzystaniu standardowego modułu Python : *pickle*. Serializacja – metoda *dump()*, deserializacja – metoda *load()*.

Dzięki tej funkcjonalności możesz zapisać i odczytać BD studentów i pracowników do i z pliku.

Zad. 4

Rozpoznaj dokładnie implementację wzorca projektowego *obserwator* w Pythonie - odpowiedź na pytania :

- jak obiekty się komunikują miedzy sobą,

- jak zaimplementować '*Javowy interfejs*' w Pythonie.

Przygotuj własny przykład demonstracyjny, wykorzystujący wzorzec projektowy obserwator.

Uwaga: ostateczny termin realizacji całej listy – 04.11.2020