PROGRAMOWANIE ZORIENTOWANE OBIEKTOWO

ĆWICZENIA DO PREZENTACJI

MODUŁ 5

Altkom Akademia S.A., materiały własne

1 KLASY I OBIEKTY

ĆWICZENIE 1.1:

Definiowanie klas i obiektów

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność:
  - definiowania klas i atrybutów instancyjnych
  - tworzenia instancji na podstawie klas
  - dostępu do atrybutów instancyjnych

CELE I ZADANIA:

- Utwórz prostą klasę reprezentującą osoby
- Zaproponuj atrybuty, jakie powinna posiadać ta klasa
- Zdefiniuj sposób tworzenia obiektów i zachowanie klas

ALGORYTM WYKONANIA:

- Utwórz klasę o nazwie *Osoba*
  - każda osoba jest opisywana przez atrybuty reprezentujące: imię (tekst), nazwisko (tekst), płeć (wartość logiczna: True – mężczyzna, False – kobieta) oraz rok urodzenia (liczba)
  - możesz też zaproponować inne atrybuty...
  - podczas tworzenia instancji wszystkie dane muszą być podane
  - zdefiniuj metodę o nazwie *ile_lat*, która obliczy aktualny wiek osoby
    - aby odczytać aktualny rok zaimportuj moduł *datetime*
    - bieżący rok zwróci wyrażenie: *datetime.date.today().year*
  - zdefiniuj metodę (o nazwie *__str__*) zwracającą opis osoby, np.: *Jan Kowalski, płeć: M, wiek: 29 lat*
- Przetestuj działanie klasy
  - utwórz osobę, podając wszystkie niezbędne dane
  - wypisz wybrane atrybuty (np. imię, a następnie nazwisko)
  - wypisz wszystkie dane osoby, podając funkcji *print* zmienną reprezentującą utworzoną instancję osoby
  - zwiększ o 1 rok urodzenia utworzonej osoby i ponownie wypisz wszystkie informacje o niej
- Czy taka konstrukcja klas chroni nas przed wprowadzeniem bezsensownych wartości (np. roku urodzenia pochodzącego z przyszłości)?
- Zobacz, jak wtedy zachowa się program

2 HERMETYZACJA

ĆWICZENIE 2.1:

Hermetyzacja

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętności:
  - hermetyzowania klas

CELE I ZADANIA:

- Dokonaj hermetyzacji klas z poprzedniego ćwiczenia 1.1 bieżącego modułu

ALGORYTM WYKONANIA:

- Zmodyfikuj klasę *Osoba* z poprzedniego ćwiczenia 1.1 bieżącego modułu
- Dokonaj jej hermetyzacji
- W tym celu atrybuty zadeklaruj jako silnie prywatne
- Modyfikacja wartości atrybutów będzie możliwa za pomocą dodatkowych metod dostępowych (np. *ustaw_imie*, *podaj_imię*, itd.)
- Zagwarantuj, że przyjmowane będą tylko wartości sensowne, tzn. imię i nazwisko nie mogą być puste, zaś rok urodzenia musi być bieżący lub przeszły
- Jak zmieni się kod testujący?
- Jakie widzisz tu utrudnienia?

ĆWICZENIE 2.2:

Wykorzystanie właściwości

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętności:
  - tworzenia i wykorzystania właściwości

CELE I ZADANIA:

- Korzystając z rozwiązania poprzedniego ćwiczenia 2.1 bieżącego modułu, zrezygnuj z hermetyzacji klasy
- Atrybuty klasy zdefiniuj jako właściwości

ALGORYTM WYKONANIA:

- Zmodyfikuj klasę (*Osoba*) z poprzedniego ćwiczenia 2.1 bieżącego modułu
- Zrezygnuj z hermetyzacji klasy
- Atrybuty zadeklaruj jako właściwości – to spowoduje, że dostęp do nich, choć wygląda jak bezpośredni, faktycznie będzie się odbywał poprzez metody dostępowe
- Przetestuj działanie programu
- Upewnij się, że inaczej niż to miało miejsce w pierwszym ćwiczeniu, teraz można kontrolować (walidować) wartości atrybutów

3 DZIEDZICZENIE

ĆWICZENIE 3.1:

Dziedziczenie – pracownicy i kierownicy zespołów

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętności:
  - definiowania klas
  - wykorzystania relacji dziedziczenia

CELE I ZADANIA:

- Utwórz klasę opisującą pracownika, zaproponuj atrybuty i zainicjuj je
- Utwórz klasę kierownika zespołu wykorzystując relację dziedziczenia (kierownik jest też pracownikiem)
- Zmień zachowanie wybranych metod, wykorzystując zachowanie oryginalne

ALGORYTM WYKONANIA:

- Utwórz klasę o nazwie *Pracownik*, reprezentującą pracownika
- Dla każdego pracownika jesteśmy w stanie określić jego imię, nazwisko i zarobki
- Utwórz i zainicjuj te atrybuty w metodzie *__init__*
- Zastanów się, którą metodę należy przedefiniować, aby dostarczyć własnego opisu obiektu (przeznaczonego dla człowieka)
- Zmień tę reprezentację tak, aby po wykonaniu instrukcji:
  ```
  p1 = Pracownik('Jan', 'Kowalski', 4_000)
  print(p1)
  ```
  otrzymać komunikat:
  ```
  [Pracownik] Jan Kowalski, zarobki: 4000
  ```

- Utwórz klasę o nazwie *KierownikZespolu*
- Klasa powinna dziedziczyć po klasie *Pracownik* (gdyż każdy kierownik, też jest pracownikiem, ale na odwrót już nie)
- Kierownik powinien posiadać te same atrybuty, co pracownik oraz dodatkowo:
  - atrybut reprezentujący listę swoich pracowników (początkowo pustą)
  - atrybut reprezentujący odpowiedzialność (inicjowany podczas tworzenia obiektu)
- Inicjalizację atrybutów wspólnych dla obu klas deleguj do klasy pracownika
- Podczas tworzenia instancji kierownika, jego zespół powinien być pusty (nie ma potrzeby podawania zespołu jako argumentu podczas tworzenia instancji kierownika, ale atrybut powinien powstać)

- Zdefiniuj w klasie kierownika metody umożliwiające poszerzenie lub zmniejszenie zespołu (np. metody: *dodaj_pracownika* i *usun_pracownika*)
- Dodatkowo dodaj metodę *kto_w_zespole*, która wypisze listę pracowników należących do zespołu danego kierownika
- Wreszcie przedefiniuj metodę zwracającą opis kierownika – powinien on być zbieżny z opisem pracownika (również wykorzystaj delegację) i poszerzony o informację o odpowiedzialności (bez informacji o zespole)
- Sprawdź działanie aplikacji

ĆWICZENIE 3.2:

Przeciążanie operatorów

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętności:
  - definiowania klas
  - przeciążania operatorów

CELE I ZADANIA:

- Utwórz klasę i przeciąż w niej wybrane operatory

ALGORYTM WYKONANIA:

- Utwórz klasę o nazwie *Osoba* – możesz ją skopiować z ćwiczenia 2.2
- Utwórz kilka instancji tej klasy
- Sprawdź zachowanie programu, gdy spróbujesz porównać instancje tych osób za pomocą operatorów: <, ==, >
- Zastanów się, które metody należy przedefiniować, aby program działał poprawnie
- Zmień zachowanie tych metod
- Operatory trzeba przeciążyć tak, aby ustalały porządek dwóch osób – o kolejności powinno najpierw decydować nazwisko, a jeśli nie będzie to rozstrzygające, to imię
- Operator porównania powinien zwrócić wartość *True*, gdy dwie osoby mają to samo imię i nazwisko (mimo tego, że mogą się różnić innymi atrybutami)
- Przykładowo:

```
Jan Kowalski <  Adam Nowak
Jan Kowalski >  Dariusz Kowalski
Jan Kowalski >  Jan Jabłoński
Jan Kowalski == Jan Kowalski
```

- Przetestuj działanie zdefiniowanych operatorów