Języki programowania – Python
Elementy programowania obiektowego

Marzena Nowakowska

Wydział Zarządzania i Modelowania Komputerowego
Politechnika Świętokrzyska

Budynek C, p. 3.21

spimn@tu.kielce.pl

Standardowa obsługa obiektu

Obiekt to zbiór danych (zmienne, własności, pola, atrybuty) i metod (funkcje), które działają na tych danych. Planem, swego rodzaju projektem (typem danych), jest **klasa**. Często na obiekt mówi się, że jest **instancją** (*instance*) klasy (przykładem jej implementacji). Można również myśleć o klasie jako szkicu (prototypie) pewnej struktury danych i operacji na nich.

Można utworzyć wiele obiektów, których schemat jest zdefiniowany przez klasę. Obiekty zdefiniowane na podstawie tej samej klasy mogą różnić się wartościami swoich właściwości. Tworzenie obiektu nosi nazwę **konkretyzacji** klasy (*instantiation*, stanowienie przykładu).

Aby posługiwać się w Pythonie (i innych językach) obiektami, najpierw muszą być zdefiniowane klasy, które opisują strukturę danych obiektu i metody, jakie na rzecz obiektu pracują.

Definicja klasy i obiektu klasy

Składnia definicji klasy:

```
class NazwaKlasy:
    instrukcja1
    instrukcja2
    …
    instrukcjaN
```

Składnia definicji obiektu:

```
obiekt = NazwaKlasy()
```

Za pomocą definicji określa się elementy **składowe klasy**, do których zalicza się między innymi:

- atrybuty klasy,
- konstruktor,
- atrybuty instancji,
- metody klasy.

Składowe klasy - 1

Atrybuty klasy to zmienne zdefiniowane w klasie, które są współdzielone przez wszystkie obiekty klasy. Dostęp do atrybutów klasy można uzyskać za pomocą nazwy klasy, a także za pomocą obiektów. Wartość atrybutu klasy pozostanie taka sama dla wszystkich obiektów, chyba że zostanie jawnie zmodyfikowana. Zmiana wartości atrybutu klasy przy użyciu nazwy klasy zmieni ją we wszystkich instancjach. Zmiana wartości atrybutu klasy za pomocą instancji nie będzie odzwierciedlona w innych instancjach lub klasie.

Konstruktor jest specjalną metodą (funkcją) klasy wywoływaną automatycznie, gdy powstaje obiekt klasy. Konstruktor ma specjalną nazwę ***__init__()*** (dwa podkreślniki z każdej strony) i specjalny, pierwszy parametr o nazwie ***self***. Służy on do definiowania atrybutów instancji i przypisywania im wartości.

Atrybuty instancji to właściwości specyficzne (określone) dla instancji klasy (czyli obiektu) – są danymi obiektu. Atrybuty instancji są definiowane w konstruktorze. Ustalenie wartości właściwości lub dostęp do właściwości **w klasie** jest realizowane poprzez notację:
self.*właściwość*

Dostęp do **składowych** (właściwości, metod) obiektu klasy **w programie** jest realizowany poprzez notację z kropką:

```
obiekt = NazwaKlasy()      # Utworzenie obiektu (instancji) danej klasy; konstruktor
obiekt.właściwość  ; obiekt.metoda()    # Odwołanie się do składowych obiektu
```

Wartości dla właściwości obiektu można przekazać poprzez parametry konstruktora (tak jak w każdej funkcji w Pythonie, w szczególności można definiować parametry domyślne).

Składowe klasy - 2

Metoda to funkcja – element składowy klasy. Metodę definiuje się w ciele klasy, za pomocą słowa kluczowego def. Klasa może mieć wiele metod. Każda metoda musi mieć pierwszy parametr, zwykle nazwany self, odwołujący się do obiektu (instancji), na rzecz którego metoda została wywołana. Jednak wywołując metodę klasy dla obiektu nie umieszcza się go na liście parametrów aktualnych (argumentów).

```
class NazwaKlasy:
    AtrybutKlasy1 = wyrażenie  # definicja atrybutu klasy; takich atrybutów może być kilka
    def __init__(self, par1k, par2k):          # konstruktor
        # parametry wartościują atrybuty dla obiektu: atr1, atr2
        # ciało konstruktora
    # definicja metody klasy bez parametrów (oprócz self) do wykonywania operacji na obiekcie
    def metoda1(self):
        # ciało metody1
    def metoda2(self, par1m, par11m):
    # definicja metody klasy z dwoma parametrami (oprócz self) do wykonywania operacji na obiekcie
        # ciało metody2
        self.metoda1()     # wywołanie metody klasy w innej metodzie tej samej klasy
        # ciało metody2
    # Koniec definicji klasy
# Utworzenie instancji (obiektu) klasy; wywołanie konstruktora z dwoma argumentami
obiekt1 = NazwaKlasy(arg1, arg11)  # parametry aktualne dla właściwości obiektu 'obiekt1'
obiekt1.metoda2(arg2, arg22)  # wywołanie metody z argumentami na rzecz utworzonego obiektu
```

Program – cz. 1: definicja własnej klasy

```
# Uwaga: w programie nie ma kontroli poprawnosci danych.
class wycieczki():
    nazwaBiura="Holiday travel"
    # liczbaW=4
    def __init__(self, nazwa, kraj, cena, maxUczest, termin):  # konstruktor
        self.nazwaW = nazwa
        self.nazwaK = kraj
        self.cenaW = cena
        self.terminW = termin
        self.maxLOsb = maxUczest
        self.lUczest=0
        self.uczestnicy=[]

    def oferta(self):
        print("Wycieczka " + self.nazwaW +" organizowana przez", self.nazwaBiura)
        print("Kraj docelowy: ", self.nazwaK)
        print("Cena za osobę: ", self.cenaW)
        print("termin wycieczki: ", self.terminW)

    def zapisy(self, osoba):
        if self.lUczest <self.maxLOsb:
            self.lUczest+=1
            self.uczestnicy.append(osoba)
            print("Osoba",osoba,"wpisana na listę")
        else:
            print("Niestety lista osób jest wyczerpana")

    def listaUcz(self):
        i=0
        for el in self.uczestnicy:
            i+=1
            print(i,"--",el)
    def przychod(self):
        return self.lUczest*self.cenaW
```

Atrybuty klasy

Atrybuty instancji

Konstruktor

Metody klasy

Zad_13-obiektWycieczka.py

Program – cz.2: implementacja klasy

```
class wycieczki():
    nazwaBiura="Holiday travel"
    def __init__(self, nazwa, kraj, cena, maxUczest, termin):          # konstruktor
            # Atrybuty obiektu: nazwaW, nazwaK, cenaW, maxLOsb, lUczest, uczestnicy
            # ciało konstruktora
    def oferta(self):
            # ciało metody
    def zapisy(self, osoba):
            # ciało metody
    def listaUcz(self):
            # ciało metody
    def przychod(self):
            # ciało metody
wycieczka1=wycieczki("'Dawna Jugosławia'", "Słowenia", 2500, 26, "2022/10/30-2022/11/07")
wycieczka1.oferta()
wycieczka1.zapisy("Czerwony Kapturek")
# ...
wycieczka1.listaUcz()
print ("Przychód z wycieczki:",wycieczka1.przychod())
print("Liczba niewykorzystanych miejsc:", wycieczka1.maxLOsb-wycieczka1.lUczest)
```

Program – listing

```
# Uwaga: w programie nie ma kontroli poprawnosci danych.
class wycieczki():
    nazwaBiura="Holiday travel"
    # liczbaW=4
    def __init__(self, nazwa, kraj, cena, maxUczest, termin):  # konstruktor
        self.nazwaW = nazwa
        self.nazwaK = kraj
        self.cenaW = cena
        self.terminW = termin
        self.maxLOsb = maxUczest
        self.lUczest=0
        self.uczestnicy=[]

    def oferta(self):
        print("Wycieczka " + self.nazwaW +" organizowana przez", self.nazwaBiura)
        print("Kraj docelowy: ", self.nazwaK)
        print("Cena za osobę: ", self.cenaW)
        print("termin wycieczki: ", self.terminW)

    def zapisy(self, osoba):
        if self.lUczest <self.maxLOsb:
            self.lUczest+=1
            self.uczestnicy.append(osoba)
            print("Osoba",osoba,"wpisana na listę")
        else:
            print("Niestety lista osób jest wyczerpana")

    def listaUcz(self):
        i=0
        for el in self.uczestnicy:
            i+=1
            print(i,"--",el)
    def przychod(self):
        return self.lUczest*self.cenaW

wycieczka1=wycieczki("'Dawna Jugosławia'", "Słowenia", 2500, 26, "2022/10/30-2022/11/07")
wycieczka1.oferta()
wycieczka1.zapisy("Czerwony Kapturek")
wycieczka1.zapisy("Zły wilk")
wycieczka1.zapisy("Królewna Śnieżka")
wycieczka1.zapisy("Książę wybawca")
wycieczka1.zapisy(input("Podaj imię i nazwisko: "))
print("Lista uczestników wycieczki")
print("==========================")
wycieczka1.listaUcz()
print ("Przychód z wycieczki:",wycieczka1.przychod())
print("Liczba niewykorzystanych miejsc:", wycieczka1.maxLOsb-wycieczka1.lUczest)

wycieczka2=wycieczki("'Meteory'", "Grecja", 3500, 14, "2023/05/10-2023/05/20")
wycieczka2.oferta()
```

Dziedziczenie

Dziedziczenie klas to mechanizm pozwalający na definiowanie nowych klas w oparciu o klasy już istniejące. Mechanizm dziedziczenia umożliwia:

- uzyskać składowe klasy bazowej,
- zmienić definicje funkcji klasy bazowej, których się nie potrzebuje,
- dodać nowe właściwości i funkcje do swojej klasy potomnej (zwanej też klasą pochodną lub podklasą).

Klasa **bazowa** (rodzicielska , nadrzędna, dziedziczona) to klasa, która dostarcza swoje narzędzia klasie **potomnej** (pochodnej, dziedziczącej, podrzędnej). Dzięki mechanizmowi dziedziczenia poszerza się lub zmienia funkcjonalność klasy bazowej.

Klasa bazowa

Definiuje składowe klasy: właściwości (dane) i metody (funkcje).



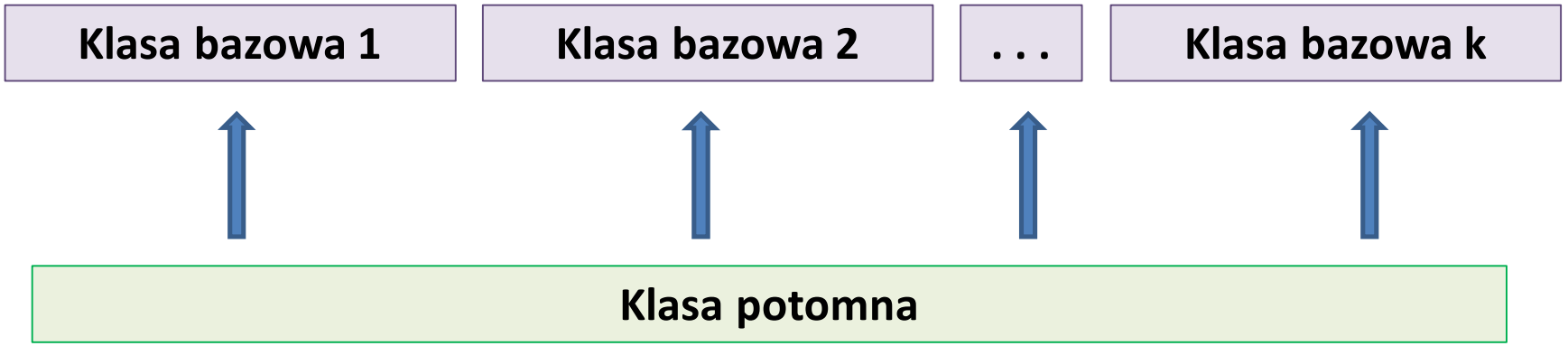
Klasa potomna

Zawiera (dziedziczy) wszystkie właściwości i metody klasy, z której się wywodzi – może się do nich odwoływać. Może modyfikować składowe odziedziczone oraz zawierać własne składowe.

Składnia klasy potomnej

```
class NazwaKlasyPotomenj(NazwaKlasyBazowej):
    instrukcja1_KP
    instrukcja2_KP
    ...
    instrukcjaN_KP
```

Dziedziczenie wielokrotne



Klasa potomna może się odwoływać do wielu klas. Może modyfikować składowe odziedziczone oraz zawierać własne składowe.

Program z dziedziczeniem klas

Zad_16-obiektyDziedziczenieWielokrotneStudent.py

```
class osoba:
    def __init__(self, par1, par2, par3):
     # Definicja konstruktora do wartościowania właściwości przy tworzeniu obiektu
     # Imię, nazwisko i data urodzenia oraz ew. inne cechy charakteryzujące osobę
    def info(self):
            print(self.imie, self.nazwisko, self.dataUr)
class studia:
    def __init__(self, par1, par2, par3):
     # Definicja konstruktora do wartościowania właściwości przy tworzeniu obiektu
     # Nazwa kierunku, liczba semestrów, profil
     def info(self):
           print("Kierunek studiów:",self.nazwa)
class student(osoba,  studia):
    def __init__(self, par11, par12, par13, par21, par22, par23, par1, par2, par3):
       osoba.__init__(self, par11, par12, par13)
       studia.__init__(self, par21, par22, par23)
       self.dyplomRok=par1; self.dyplomTytul=par2; self.dyplomOcena=par3
    def info(self):
       osoba.info(self)
       studia.info(self)
       print("Rok ukończenia studiów:", self.dyplomRok, "Ocena na dyplomie:",  self.dyplomOcena)

stdnt1=student("Janko", "Muzykant", "1998-10-12", "Zarządzanie biznesowe",
                                6, "praktyczny", 2022, "Restrukturyzacja MŚP", 4.5)
stdnt1.info()
```

Program – listing

```
class osoba:
    def __init__(self, par1, par2, par3):
# definicja konstruktora do wartościowania właściwości przy tworzeniu obiektu
        self.imie = par1
        self.nazwisko = par2
        self.dataUr = par3
# Inne cechy charakteryzujące osobę
    def info(self):
        print(self.imie, self.nazwisko, self.dataUr)

class studia:
    def __init__(self, par1, par2, par3):
# definicja konstruktora do wartościowania właściwości przy tworzeniu obiektu
        self.nazwa = par1   # kierunke studiów
        self.lSem = par2# liczba semestrów
        self.profil = par3      # profil: akademicki, praktyczny
    def info(self):
        print("Kierunek studiów:",self.nazwa)
class student(osoba,  studia):
    def __init__(self, par11, par12, par13, par21, par22, par23, par1, par2, par3):
        osoba.__init__(self, par11, par12, par13)
        studia.__init__(self, par21, par22, par23)
        self.dyplomRok=par1
        self.dyplomTytul=par2
        self.dyplomOcena=par3
    def info(self):
        osoba.info(self)
        studia.info(self)
        print("Rok ukończenia studiów:", self.dyplomRok, "Ocena na dyplomie:",
              self.dyplomOcena)

stdnt1=student("Janko", "Muzykant", "1998-10-12", "Zarządzanie biznesowe",
                6, "praktyczny", 2022, "Restrukturyzacja MŚP", 4.5)
stdnt1.info()
```