

# Programowanie obiektowe

## Zadania lab3

Piotr Błaszyński

1 kwietnia 2022

Klasy z poszczególnych zadań zapisywać w oddzielnych plikach, jeżeli w zadaniu jest kilka klas, to można je (przynajmniej na razie) też przechowywać w jednym pliku, jedna po drugiej. Wszystkie klasy pochodne przetestować (niektóre bazowe wg uznania), tworząc kilka obiektów klasy i uruchamiając każdą z dostępnych metod przynajmniej raz.

1. Przygotować klasę `Kształt` - i klasę która pozwala na zmianę rozmiarów - z tych dwóch klas odziedziczyć klasy: `Kwadrat`, `koło`, w tych klasach zaimplementować zmianę rozmiarów, zastanowić się jakie byłyby najlepsze parametry takiej metody,
2. Zaimplementować klasę `Mapa`, która składa się z obiektów klasy `Pokój` (pokój może być zamurowany - zmienna boolowska - wtedy do niego nie można wejść). W każdym niezamurowanym pokoju znajduje się obiekt klasy bazowej `Przedmiot` - ma ona 2 pochodne klasy: `wartościowy`, `śmieć`. Dorobić klasę `gracz` z listą przedmiotów, będziemy przesuwac gracza przy pomocy input (uwaga - w IDLE nie działa funkcja `msvcrt.getch()`) i po wejściu do pokoju podnosi przedmiot. Po wpisaniu "wypisz" wypisywana jest lista przedmiotów. W wersji zaawansowanej mapę (z oznaczeniem przedmiotów) można wczytywać z pliku.
3. Przygotować klasę do przechowywania `Komentarzy` (jeden obiekt - jeden komentarz napisany przez jednego użytkownika) - użytkownicy o różnych możliwościach (istnieje inna metoda do sprawdzania uprawnień - klasa bazowa `użytkownik` - klasy pochodne: `zwykły` i `administrator` (ma możliwość edycji nie swoich komentarzy)).
4. Przygotować klasę bazową reprezentującą `Figury szachowe` - sprawdzenie prawidłowości wykonania ruchu na pustej szachownicy. Po klasie `Figura`

(zgodnie z listą poniżej) dziedziczą klasy do weryfikacji poszczególnych rodzajów figur. W `__init__` klasy bazowej podajemy pole startowe (numeracja pól od a1 do h8 - linie pionowe oznaczane literami od a do h, linie poziome liczbami od 1 do 8). Później dla obiektu są dwie metody: `czy_mozna_ruch` i `wykonaj_ruch` (to tylko propozycje nazwy), obydwie wywoływane z polem docelowym jako parametr. Druga z metod zmienia pole na którym znajduje się figura.

| Figura  | reguła poruszania                                                                |
|---------|----------------------------------------------------------------------------------|
| Pion    | do przodu o jedno pole,<br>dwa pola w pierwszym ruchu                            |
| Wieża   | Tylko w pionie i poziomie                                                        |
| Skoczek | Dwa pola w pionie i jedno w poziomie<br>lub dwa pola w poziomie i jedno w pionie |
| Goniec  | Tylko po skosie                                                                  |
| Hetman  | Po skosie, w pionie i poziomie                                                   |
| Król    | Po jednym polu po skosie, w pionie i poziomie                                    |

Dla przypomnienia, przykładowa klasa `Reka`, dziedzicząca z `Organ` i korzystanie z niej:

```
class Organ:
    pass
class Reka (Organ):
    def __init__(self , ktora):
        self.ktora=ktora
    def pokaz(self):
        print(self.ktora)

prawa = Reka("PRAWA")
lewa = Reka("LEWA")

prawa.pokaz()
lewa.pokaz()
```

W poniższych przykładach 'w' oznacza zapis do, a 'r' odczyt z pliku. Lepszy sposób na odczyt pliku (nie trzeba m.in. pamiętać o close):

```
with open('plik.txt', 'r') as plik:
    tekst = plik.read()
```

Lepszy sposób na zapis do pliku (nie trzeba m.in. pamiętać o close):

```
with open('plik.txt', 'w') as plik:
    plik.write("tresc")
```

Sposób na zapis do pliku kilku linii oddzielonych znakami końca linii (wewnątrz bloku with może być też użyta pętla):

```
with open('plik.txt', 'w') as plik:  
    plik.write(str(linia_a)+'\n')  
    plik.write(str(linia_b)+'\n')  
    plik.write(str(linia_c)+'\n')
```