# Programowanie Obiektowe
# Instrukcja do laboratorium 5

Marzec 2023

## 1 Zasady oceniania

| Zadanie | Ocena |
|---|---|
| Transport (3.1) | 3,0 |
| Gracz i obiekty (3.2) | 4,0 |
| Równania (3.3) | 5,0 |

W przypadku nie oddania zadania w terminie (tydzień po zajęciach), uzyskana ocena jest zmniejszana o 0,5 za każdy tydzień opóźnienia.

Plik (wkleić cały kod do jednego pliku, podpisać poszczególne zadania za pomocą komentarzy) ze zrobionymi zadaniami proszę przesłać na platformie Moodle. Plik musi mieć nazwe `numeralbumu_lab5.py`. **Plik musi być plikiem tekstowym z rozszerzeniem `.py`. Pliki w innych formatach niż `.py` nie będą sprawdzane**.

**UWAGA:** Termin oddania zadania jest ustawiony w systemie moodle. W przypadku nie oddania zadania w terminie, uzyskana ocena będzie zmniejszana o 0,5 za każdy zaczęty tydzień opóźnienia. Zadania oddawane później niż miesiąc po terminie ustawionym na moodle są oddawane i rozliczane w trybie indywidualnym na zajęciach lub po umówieniu się z prowadzącym.

**UWAGA:** W przypadku wysłania zadania w formie niezgodnej z opisem w instrukcji prowadzący zastrzega prawo do wystawienia oceny negatywnej za taką pracę. Przykład: wysłanie `.zip` lub `.pdf` tam, gdzie był wymagany plik tekstowy z rozszerzeniem `.py`.

## 2 Materiał pomocniczy

### 2.1 Dziedziczenie konstruktora

Przykłady z wyjaśnieniem metody **super()** oraz logiką **przesłaniania klas** znajdują się na poniższych linkach:

- Dokumentacja Python
- Programiz
- w3schools
- Real Python
- Grzegorz Jagiella (Uniwersytet Wrocławski)

### 2.2 Klasy abstrakcyjne

Przykłady z wyjaśnieniem logiki klas abstrakcyjnych **(moduł abc, metoda magiczna `@abstractmethod`)** znajdują się na poniższych linkach:

- OSCAR SIERRA PROJECT
- Moduł ABC
- Przykłady Python

### 2.3 Polimorfizm

Przykłady z wyjaśnieniem podstawowych zagadnień dotyczących polimorfizmu **(metody *type()* oraz *isinstance()*)** znajdują się na poniższych linkach:

- Programiz
- Geeks For Geeks
- Kodołamacz

### 2.4 Wielokrotne dziedziczenie

Przykłady z wyjaśnieniem podstawowych zagadnień dotyczących wielokrotnego dziedziczenia znajdują się na poniższych linkach:

- Programiz
- Geeks For Geeks
- Python-course
- Dokumentacja Python

## 3 Zadania do wykonania

### 3.1 Transport

Zaimplementuj hierarchię klas w języku Python dla firmy logistycznej. Klasa abstrakcyjna będzie reprezentować transport, a klasy pochodne reprezentują różne środki transportu, takie jak samolot, statek i ciężarówka.

Klasa abstrakcyjna `Transport` powinna mieć trzy pola: `start`, `koniec` i `ładunek`, które będą reprezentować miejsce rozpoczęcia transportu, miejsce docelowe i ładunek. Klasa abstrakcyjna powinna mieć również cztery metody:

- `transportuj` - abstrakcyjna metoda, która będzie zaimplementowana w klasach pochodnych (posiada argument `nowy_koniec`)
- `get_start` - zwraca miejsce rozpoczęcia transportu
- `get_koniec` - zwraca miejsce docelowe transportu
- `get_ladunek` - zwraca informacje o ładunku

Klasy pochodne `Samolot`, `Statek` i `Ciezarowka` powinny dziedziczyć po klasie abstrakcyjnej `Transport` i dodawać dwa dodatkowe pola:

- dla `Samolot` - `ilosc_pasazerow`, `ilosc_bagazy`
- dla `Statek` - `rodzaj_ladunku`, `ilosc_kontenerow`
- dla `Ciezarowka` - `ilosc_palet`, `typ_ladunku`

Każda klasa pochodna powinna mieć metodę `transportuj`, która będzie implementowała transport dla danego środka transportu. Metoda ta powinna również wykorzystywać pola dziedziczone z klasy abstrakcyjnej. Metoda `transportuj` powinna realizować przemieszczenie ładunku z miejsca startu do miejsca docelowego, czyli musi zmieniać wartość pola `start` na wartość pola `koniec`, a wartość pola `koniec` zamieniać na wartość argumentu podanego do metody `transportuj`.
W klasie `Samolot`, metoda `transportuj` ma wyglądać następująco:

- Wyświetl komunikat, że samolot startuje z miejsca `start` do miejsca `koniec`.
- Wyświetl informacje o ilości pasażerów i bagażu.
- Przemieszczenie samolotu z miejsca start do miejsca koniec.

W klasie `Statek`, metoda `transportuj` ma wyglądać następująco:

- Wyświetl komunikat, że statek wypływa z portu `start` do portu `koniec`.
- Wyświetl informacje o rodzaju ładunku i ilości kontenerów.
- Przemieszczenie statku z portu `start` do portu `koniec`

W klasie `Ciezarowka`, metoda `transportuj` może wyglądać następująco:

- Wyświetl komunikat, że ciężarówka rusza z miejsca `start` do miejsca `koniec`.
- Wyświetl informacje o ilości palet i typie ładunku.
- Przemieszczenie ciężarówki z miejsca `start` do miejsca `koniec`.

W każdej klasie pochodnej metoda `transportuj` powinna wykorzystywać metody dziedziczone z klasy abstrakcyjnej w celu pobrania informacji o miejscu startu, miejscu docelowym i ładunku. Klasy powinny również zawierać metodę magiczną `__str__` i `__repr__` w celu wyświetlenia informacji o obiekcie. Niech w przypadku tego zadania metoda `__repr__` wywołuje zawsze metodę `__str__`.

Listing 1: Przykład

```
>>> # tworzenie obiektu statku
>>> statek = Statek("Gdańsk", "New York", "kontenery", "Towar A", 100)
>>> # wywołanie metody transportuj na obiekcie statku
>>> statek.transportuj("Hamburg")
Statek wypływa z Gdańsk do New York.
Rodzaj ładunku: kontenery, ilość kontenerów: 100.
Statek dotarł do New York
>>> # wyświetlenie informacji o statku
>>> print(statek)
Statek(start=New York, koniec=Hamburg, ladunek=kontenery, rodzaj_ladunku=Towar A,
    ilosc_kontenerow=100)
```

## 3.2 Gracz i potwory

Utworzyć klasę abstrakcyjną `GameObject`. Ta klasa ma posiadać konstruktor przyjmujący jako jedyny argument liczby punktów zdrowia obiektu. Ta klasa ma posiadać także metodę sprawdzającą czy obiekt "żyje"(zwracamy `True` jeżeli punkty zdrowia są większe od zera), oraz metodę abstrakcyjną `.interact()` która ma przyjmować jeden argument (później będzie wyjaśnione jaki).

Po tej klasie muszą dziedziczyć trzy inne klasy: `Player`, `Monster`, oraz `Door`, które reprezentują odpowiednio gracza, potwora i drzwi.

Żadna z utworzonych klas nie nadpisuje konstruktora, ponieważ będzie używany konstruktor z klasy bazowej.

W klasie `Player` metoda `.interact()` musi być nadpisana, lecz nie musi ona posiadać implementacji (jeżeli użyć dekorator `@abstractmethod` i nie nadpisać tej metody w klasie odziedziczonej to nie będziemy mogli utworzyć obiektu tej klasy). Zakładamy, że jako argument do dej metody będzie zawsze trafiał obiekt klasy `Player`.

W klasie `Door` metoda `.interact()` musi wyświetlać informacje o tym że gracz przeszedł przez drzwi.

W klasie `Monster` metoda `.interact()` musi zmniejszyć liczbę punktów zdrowie obiektu podanego jako argument (czyli gracza) o 10, następnie ustawić punkty zdrowia tego potwora na 0 i wyświetlić informacje o tym że potwór został zabity przez gracza.

Po utworzeniu wyżej opisanych klas, należy napisać kod symulujący prostą grę: tworzymy obiekt gracza (obiekt klasy `Player`), następnie tworzymy listę i losowo wypełniamy ją zadaną liczbą obiektów klas `Monster` albo `Door` (używamy warunek który pozwoli wybrać losowo obiekt której klasy zostanie dodany do listy).

Następnie, należy napisać pętlę, która będzie przechodziła po obiektach z tej listy i wywoływać metodę `.interact()` na obiektach z tej listy. Podajemy wcześniej utworzony obiekt gracza jako argument do tej metody (`obj.inteact(player)`). Na końcu każdej iteracji sprawdzamy czy gracz jeszcze żyje (metoda do tego była zaimplementowana w klasie bazowej). W przypadku gdyby grać zmarł, wyświetlamy informacje o tym i przerywamy działania pętli.

Na listingu niżej jest pokazany przykładowy wynik działania takiego programu, założenia wstępne są takie: 10 obiektów na liście, 50 punktów zdrowia u gracza, oraz 70% szansa że na listę zostanie dodany potwór.

Listing 2: Przykład działania programu.

```
Gracz przeszedł przez drzwi.
Gracz przeszedł przez drzwi.
Gracz zabił potwora.
Gracz zabił potwora.
Gracz zabił potwora.
Gracz przeszedł przez drzwi.
Gracz zabił potwora.
Gracz przeszedł przez drzwi.
Gracz zabił potwora.
Gracz został zabity!
```

### 3.3 Równania

Utwórz klasę abstrakcyjną reprezentującą równanie o zadanych współczynnikach. Klasa abstrakcyjna równania musi posiadać konstruktor, przyjmujący listę współczynników równania, oraz metodę `.solve()` która musi rozwiązywać równanie (nie będzie posiadała żadnej implementacji w klasie abstrakcyjnej).

Następnie stwórz dwie klasy, dziedziczące po klasie równania. Te klasy muszą reprezentować równanie liniowe i kwadratowe. Należy nadpisać i zaimplementować metodę `.solve()` która będzie rozwiązywać równanie o współczynnikach podanych jako argument konstruktora. Należy także nadpisać konstruktor klasy bazowej w celu sprawdzenia czy podana liczba współczynników odpowiada temu równaniu (sprawdzamy czy lista posiada dwa współczynniki dla równania liniowego albo czy posiada trzy współczynniki w przypadku równania kwadratowego).

Listing 3: Przykład

```
>>> eq = LinearEquation([2, 0])
>>> eq.solve()
x = 0
>>> eq1 = LinearEquation([0, 2])
>>> eq1.solve()
Brak rozwiązań.
>>> eq2 = QuadraticEquation([1, -5, 6])
>>> eq2.solve()
x1 = 2, x2 = 3
```