Tytuł ćwiczenia: Jednostka 5 - Paradygmat obiektowy (cz. 2)
Autor: Joanna Dagil
Grupa: TCH-1
Data: 04.11.2025

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z dziedziczeniem, polimorfizmem i interfejsami w paradygmacie obiektowym.

2 Przebieg ćwiczenia

W tym ćwiczeniu posłużę się językiem Python w edytorze Visual Studio Code. W celu wykonania zadań zapoznaję się z tutorialami, a następnie przechodzę do stworzenia plików:

pracownicy.py

```python
class Pracownik:
    def pensja(self):
       return 0

class Programista(Pracownik):
    def pensja(self): # nadpisanie metody
        return 10000

class Kierownik(Pracownik):
    def pensja(self): # nadpisanie metody
        return 15000

if __name__ == "__main__":
    pracownicy = [ Programista(), Programista(), Kierownik()]
    for p in pracownicy:
        print(p.pensja())
```

figury.py

```python
from abc import ABC, abstractmethod
from math import pi

class Figura(ABC):
    @abstractmethod # stworzenie metody abstrakcyjnej
    def pole(self):
       pass

class Kolo(Figura):
    def __init__(self, r):
        self.r = r
    def pole(self): # nadpisanie metody abstrakcyjnej
        return pi * self.r ** 2

class Prostokat(Figura):
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def pole(self): # nadpisanie metody abstrakcyjnej
        return self.a * self.b

class Trojkat(Figura):
    def __init__(self, a, h):
        self.a = a
        self.h = h
    def pole(self): # nadpisanie metody abstrakcyjnej
        return self.a * self.h / 2

if __name__ == "__main__":
    figury = [ Kolo(5), Prostokat(4, 6), Trojkat(4, 5), Kolo(3) ]
    for f in figury:
        print(f.pole())
```

pojazdy.py

```
class Pojazd:
    def __init__(self, marka):
        self.marka = marka
    def info(self):
        print(f"Pojazd marki {self.marka}")
    def uruchom(self):
        print("Uruchamiam pojazd...")

class PojazdSilnikowy(Pojazd):
    def uruchom(self): # przesłonięcie metody z klasy bazowej
        print("Uruchamiam pojazd silnikowy...")

class Motocykl(PojazdSilnikowy):
    def __init__(self, marka, pojemnosc):
        super().__init__(marka)
        self.pojemnosc = pojemnosc
    def info(self): # przesłonięcie metody z klasy bazowej
        print(f"Motocykl: {self.marka}, pojemność: {self.pojemnosc}cc")

class Samochod(PojazdSilnikowy):
    def __init__(self, marka, liczba_drzwi):
        super().__init__(marka)
        self.liczba_drzwi = liczba_drzwi
    def info(self): # przesłonięcie metody z klasy bazowej
        print(f"Samochód: {self.marka}, drzwi: {self.liczba_drzwi}")

class Rower(Pojazd):
    def __init__(self, marka, typ):
        super().__init__(marka)
        self.typ = typ
    def info(self): # przesłonięcie metody z klasy bazowej
        print(f"Rower {self.marka}, typ: {self.typ}")

if __name__ == "__main__":
    pojazdy = [ Samochod("Toyota", 5), Rower("Merida", "górski"), Samochod("BMW", 3),
    Motocykl("Yamaha", 600) ]
    for p in pojazdy:
        p.info()
        p.uruchom()
```

Program uruchamiam poleceniem

```
python {nazwa}.py
```

3 Wyniki działania

Dla **pracownicy.py**:

```
10000
10000
15000
```

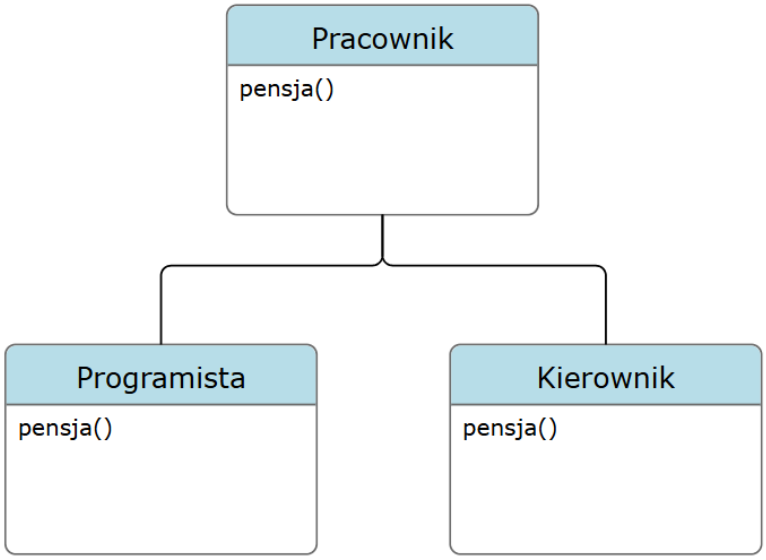
Dla **figury.py**:

```
78.53981633974483
24
10.0
28.274333882308138
```

Dla **pojazdy.py**:

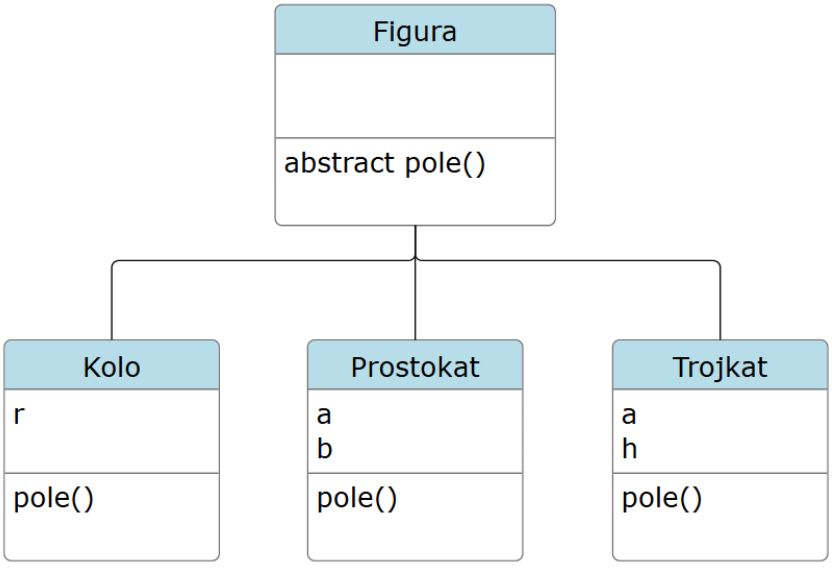
```
Samochód: Toyota, drzwi: 5
Uruchamiam pojazd silnikowy...
Rower Merida, typ: górski
Uruchamiam pojazd...
Samochód: BMW, drzwi: 3
Uruchamiam pojazd silnikowy...
Motocykl: Yamaha, pojemność: 600cc
Uruchamiam pojazd silnikowy...
```

4 Diagram hierarchii klas i opis polimorfizmu



Rysunek 1: Diagram dla zadania 1

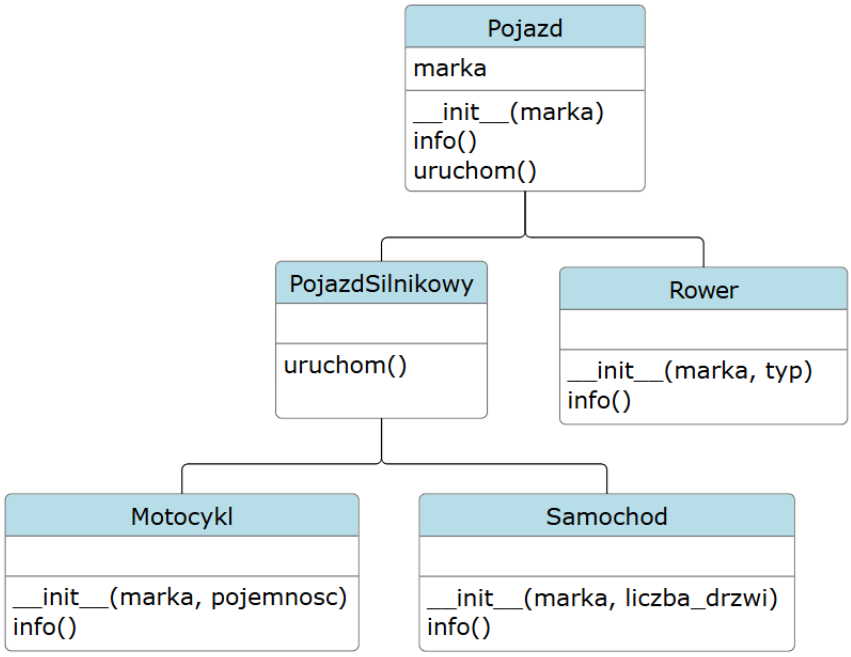
Polimorfizm w zadaniu 1. umożliwia traktowanie obiektów klas zarówno Programisty jak i Kierownika, jako obiektów typu Pracownik. Używamy tego do stworzenia listy pracownicy, w której mogą znajdować się zarówno Programiści, jak i Kierownicy. Możemy wtedy wywoływać tak samo nazwaną funkcję, ale otrzymywać inne wyniki dla każdego rodzaju obiektu.



Rysunek 2: Diagram dla zadania 2

W tym przypadku polimorfizm ponownie pozwala nam na przechowywanie różnych rodzajów figur w jednej liście. Podobnie każda figura ma posiadać metodę zwracającą jej pole, natomiast mamy

dodatkowo zabezpieczenie, że nie możliwe jest stworzenie niewspecyfikowanego rodzaju figury i żądanie jej pola - ma to sens z perspektywy geometrii.



Rysunek 3: Diagram dla zadania 3

Tutaj zastosowane jest dziedziczenie wielopoziomowe - pomiędzy klasą bazową, a klasami końcowymi istnieją także klasy pośrednie. Umożliwia to jeszcze optymalniejsze pogrupowanie klas i zmniejszenie powtarzalności kodu - gdyby klasa PojazdSilnikowy nie istniała, musielibyśmy nadpisać metodę uruchom() i w klasie Motocykl i w klasie Samochod.

5 Wnioski

Programowanie obiektowe posługuje się polimorfizmem, a więc umożliwia nadpisywanie metod z klas, które są dziedziczone, przez klasy dziedziczące. Więc różne obiekty mogą reagować różnie, mimo, że pochodzą z tej samej klasy bazowej.