Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 4 – Paradygmat obiektowy (cz. 1) – podstawy

Kierunek: Informatyka, semestr 5
Prowadzący: dr inż. Krzysztof Malczewski

1. Cel i zakres zajęć

Celem zajęć jest:

- zrozumienie idei **paradygmatu obiektowego**,
- poznanie podstawowych pojęć: **klasa, obiekt, atrybut, metoda**,
- umiejętność tworzenia klas i obiektów w języku wysokiego poziomu (Python / C++),
- implementacja prostych modeli rzeczywistych w stylu obiektowym,
- porównanie podejścia obiektowego z wcześniejszymi (imperatywnym, proceduralnym, modularnym).

Zakres tematyczny:

- definicja i główne założenia paradygmatu obiektowego,
- klasy i obiekty, enkapsulacja, stan i zachowanie,
- konstruktory i inicjalizacja obiektów,
- proste przykłady w Pythonie i C++,
- pierwsze zadania obiektowe.

---

2. Wprowadzenie teoretyczne

2.1 Geneza paradygmatu obiektowego

Paradygmat obiektowy (Object-Oriented Programming – OOP) pojawił się jako odpowiedź na rosnącą złożoność oprogramowania. Umożliwia on **modelowanie rzeczywistości w sposób naturalny**, przez odwzorowanie obiektów rzeczywistych jako **obiektów programistycznych**.

2.2 Kluczowe pojęcia

- **Klasa** — szablon definiujący strukturę i zachowanie obiektów: atrybuty (dane) i metody (funkcje operujące na danych).
- **Obiekt** — konkretny egzemplarz klasy, posiadający własny stan.
- **Atrybut** — zmienna wewnątrz klasy przechowująca stan obiektu.
- **Metoda** — funkcja wewnątrz klasy operująca na stanie obiektu.
- **Konstruktor** — specjalna metoda służąca do tworzenia i inicjalizacji obiektów.
- **Enkapsulacja** — ukrycie szczegółów implementacji i kontrola dostępu do danych przez metody.

2.3 Cechy OOP

- **Abstrakcja** – reprezentowanie tylko istotnych cech obiektu.
- **Enkapsulacja** – ukrywanie szczegółów wewnętrznych, udostępnianie interfejsów.
- **Modularność** – program składa się z niezależnych klas.
- **Reużywalność** – klasy mogą być wykorzystywane wielokrotnie.

- **Łatwość rozszerzania** – klasy można rozbudowywać i łączyć.

---

3. Tutorial 1 – Klasy i obiekty (Python)

3.1 Definiowanie klasy

```
class Samochod:
    def __init__(self, marka, rocznik, przebieg=0):
        self.marka = marka          # atrybut
        self.rocznik = rocznik
        self.przebieg = przebieg

    def jedz(self, km):
        """Zwiększa przebieg o podaną liczbę kilometrów."""
        self.przebieg += km

    def info(self):
        print(f"{self.marka} ({self.rocznik}) - przebieg: {self.przebieg}
km")
```

3.2 Tworzenie obiektów i korzystanie z metod

```
if __name__ == "__main__":
    s1 = Samochod("Toyota", 2015)
    s2 = Samochod("BMW", 2018, 30000)

    s1.jedz(150)
    s2.jedz(500)

    s1.info()
    s2.info()
```

Wynik przykładowy:

```
Toyota (2015) - przebieg: 150 km
BMW (2018) - przebieg: 30500 km
```

3.3 Konstruktory i stan obiektu

- Konstruktor `__init__` ustawia początkowy stan obiektu.
- Atrybuty należą do konkretnego egzemplarza, nie do klasy.
- Każdy obiekt ma niezależny stan.

---

4. Tutorial 2 – Klasy w C++ (opcjonalnie)

Dla studentów zaznajomionych z C++.

```
#include <iostream>
#include <string>
using namespace std;

class Samochod {
private:
    string marka;
    int rocznik;
    double przebieg;
public:
    Samochod(string m, int r, double p = 0)
        : marka(m), rocznik(r), przebieg(p) {}

    void jedz(double km) { przebieg += km; }
    void info() const {
        cout << marka << " (" << rocznik << ") - przebieg: "
             << przebieg << " km" << endl;
    }
};

int main() {
    Samochod s1("Toyota", 2015);
    Samochod s2("BMW", 2018, 30000);
    s1.jedz(150);
    s2.jedz(500);
    s1.info();
    s2.info();
    return 0;
}
```

---

5. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Zdefiniuj klasę `Student`, która zawiera:

- atrybuty: `imie`, `nazwisko`, `oceny` (lista),
- metodę `dodaj_ocene(ocena)`,
- metodę `srednia()`,
- metodę `info()` wypisującą dane studenta i średnią.

Stwórz kilka obiektów `Student` i zaprezentuj działanie metod.

Zadanie 2 (obowiązkowe)

Zdefiniuj klasę `Prostokat`, która posiada:

- atrybuty: `szerokosc`, `wysokosc`,

- metody: `pole()`, `obwod()`, `skaluj(faktor)`.

Napisz program, który tworzy kilka prostokątów, skaluje je i wypisuje wyniki.

Zadanie 3 (dodatkowe)

Zaprojektuj klasę `BankKonto`, z metodami: `wplac()`, `wyplac()`, `saldo()`. Uwzględnij kontrolę, aby nie można było wypłacić więcej niż saldo.

---

6. Pytania kontrolne

1. Czym różni się klasa od obiektu?
2. Co to jest enkapsulacja i dlaczego jest ważna?
3. Jaką rolę pełni konstruktor?
4. Co się stanie, jeśli utworzymy dwa różne obiekty tej samej klasy?
5. W jaki sposób OOP poprawia organizację kodu w porównaniu do podejść proceduralnych?

---

7. Checklisty

Projekt OOP

- Klasy mają jasno określone atrybuty i metody.
- Obiekty są tworzone i używane poprawnie.
- Konstruktor inicjalizuje stan.
- Kod nie używa zmiennych globalnych do przechowywania stanu.

---

8. Wzór sprawozdania

Dodatkowo:

- Opisz **strukturę klas** (diagram lub opis).
- Wyjaśnij różnice między Twoim programem obiektowym a wersją proceduralną.
- Zawrzyj zrzuty ekranu z działania programów i testów.

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć podstawowe pojęcia paradygmatu obiektowego,
- potrafić projektować i implementować klasy z atrybutami i metodami,
- umieć modelować proste byty rzeczywiste jako obiekty,
- rozumieć różnicę między podejściem obiektowym a wcześniejszymi.