Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 5 – Paradygmat obiektowy (cz. 2): Dziedziczenie i polimorfizm

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

1. Cel i zakres zajęć

Celem zajęć jest:

- zrozumienie pojęć **dziedziczenia**, **polimorfizmu** i **interfejsów** w programowaniu obiektowym,
- nauczenie się tworzenia hierarchii klas i wykorzystania klas bazowych oraz pochodnych,
- zrozumienie mechanizmu przesłaniania metod,
- wykorzystanie polimorfizmu do pisania elastycznego kodu.

Zakres tematyczny:

- dziedziczenie klas, klasy bazowe i pochodne,
- nadpisywanie i rozszerzanie metod,
- wywoływanie metod klasy bazowej,
- interfejsy i klasy abstrakcyjne,
- polimorfizm (dynamiczne wiązanie metod).

---

2. Wprowadzenie teoretyczne

2.1 Dziedziczenie

Dziedziczenie pozwala tworzyć **nowe klasy** w oparciu o już istniejące.

- **Klasa bazowa** definiuje ogólne zachowanie.
- **Klasa pochodna** dziedziczy atrybuty i metody, może je rozszerzać lub nadpisywać.

Zalety:

- Reużywalność kodu,
- Hierarchia i porządek,
- Możliwość rozszerzania bez modyfikowania oryginalnych klas.

2.2 Polimorfizm

Polimorfizm pozwala różnym klasom **reagować na to samo wywołanie metod w różny sposób**.
Np. `obiekt.rysuj()` może zachowywać się inaczej w zależności od rzeczywistego typu obiektu.

Wyróżniamy:

- **Polimorfizm dziedziczenia** – poprzez nadpisywanie metod.
- **Polimorfizm interfejsów** – poprzez wspólny zestaw metod w różnych klasach.

2.3 Interfejsy i klasy abstrakcyjne

- **Interfejs** – zbiór metod, które klasa musi zaimplementować.
- **Klasa abstrakcyjna** – klasa, której nie można utworzyć, zawiera metody abstrakcyjne.

---

3. Tutorial 1 – Dziedziczenie i polimorfizm (Python)

3.1 Klasa bazowa i pochodne

```
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 Samochod(Pojazd):
    def __init__(self, marka, liczba_drzwi):
        super().__init__(marka)
        self.liczba_drzwi = liczba_drzwi

    def info(self):
        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):
        print(f"Rower {self.marka}, typ: {self.typ}")
```

3.2 Polimorfizm – wspólne wywołanie

```
pojazdy = [
    Samochod("Toyota", 5),
    Rower("Merida", "górski"),
    Samochod("BMW", 3)
]
```

```
for p in pojazdy:
    p.info()       # różne zachowanie dla różnych klas
    p.uruchom()    # wspólna metoda bazowa
```

Wynik przykładowy:

```
Samochód: Toyota, drzwi: 5
Uruchamiam pojazd...
Rower Merida, typ: górski
Uruchamiam pojazd...
Samochód: BMW, drzwi: 3
Uruchamiam pojazd...
```

3.3 Klasy abstrakcyjne (opcjonalnie, zaawansowane)

```
from abc import ABC, abstractmethod

class Figura(ABC):
    @abstractmethod
    def pole(self):
        pass

class Kolo(Figura):
    def __init__(self, r):
        self.r = r
    def pole(self):
        from math import pi
        return pi * self.r**2

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

figury = [Kolo(3), Prostokat(4, 5)]
for f in figury:
    print(f"Pole: {f.pole()}")
```

---

4. Tutorial 2 – Dziedziczenie w C++ (dla chętnych)

```
#include <iostream>
#include <vector>
#include <memory>
using namespace std;

class Pojazd {
protected:
    string marka;
public:
    Pojazd(string m): marka(m) {}
```

```
    virtual ~Pojazd() {}
    virtual void info() const {
        cout << "Pojazd marki " << marka << endl;
    }
    void uruchom() const {
        cout << "Uruchamiam pojazd..." << endl;
    }
};

class Samochod : public Pojazd {
    int drzwi;
public:
    Samochod(string m, int d): Pojazd(m), drzwi(d) {}
    void info() const override {
        cout << "Samochód: " << marka << ", drzwi: " << drzwi << endl;
    }
};

class Rower : public Pojazd {
    string typ;
public:
    Rower(string m, string t): Pojazd(m), typ(t) {}
    void info() const override {
        cout << "Rower " << marka << ", typ: " << typ << endl;
    }
};

int main() {
    vector<shared_ptr<Pojazd>> pojazdy;
    pojazdy.push_back(make_shared<Samochod>("Toyota", 5));
    pojazdy.push_back(make_shared<Rower>("Merida", "górski"));

    for (auto &p : pojazdy) {
        p->info();
        p->uruchom();
    }
    return 0;
}
```

---

5. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Zdefiniuj hierarchię klas:

- Klasa bazowa `Pracownik` z metodą `pensja()`.
- Klasy pochodne `Programista` i `Kierownik`.
- Nadpisz metodę `pensja()` w każdej klasie, aby zwracała inną wartość.
- Stwórz listę obiektów różnych klas i wypisz ich pensje polimorficznie.

Zadanie 2 (obowiązkowe)

Zaprojektuj klasę abstrakcyjną `Figura` i klasy pochodne: `Kolo`, `Prostokat`, `Trojkat`.
Zaimplementuj metodę abstrakcyjną `pole()`.
Stwórz listę figur i wypisz ich pola w jednej pętli.

Zadanie 3 (dodatkowe)

Rozbuduj hierarchię `Pojazd` o klasę pośrednią `PojazdSilnikowy` i kolejną pochodną `Motocykl`.
Pokaż działanie metod bazowych i przesłanianych.

---

6. Pytania kontrolne

1. Czym jest dziedziczenie i jakie daje korzyści?
2. Czym różni się nadpisywanie od przeciążania metod?
3. Na czym polega polimorfizm?
4. Co to jest klasa abstrakcyjna?
5. Jak wygląda polimorfizm w Pythonie i w C++?

---

7. Checklisty

Dziedziczenie i polimorfizm

- Utworzono klasę bazową i pochodne.
- Zastosowano `super()` lub wywołanie konstruktora bazowego.
- Przesłonięto metody w klasach pochodnych.
- Polimorficzne wywołania działają poprawnie.
- Kod jest czytelny i zgodny z zasadami OOP.

---

8. Wzór sprawozdania

Dodatkowo:

- Diagram hierarchii klas.
- Wyjaśnienie, jak działa polimorfizm w twoim przykładzie.
- Fragmenty kodu z zaznaczonymi przesłoniętymi metodami.

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć dziedziczenie, polimorfizm i interfejsy,
- potrafić projektować hierarchie klas,
- umieć tworzyć klasy bazowe, pochodne i abstrakcyjne,
- stosować polimorfizm do pisania elastycznego kodu.