Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 12 – Paradygmat aspektowy (AOP – Aspect-Oriented Programming)

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

1. Cel i zakres zajęć

Celem zajęć jest:

- poznanie **paradygmatu aspektowego** i jego roli w projektowaniu oprogramowania,
- zrozumienie pojęć: **aspekt, punkt zaczepienia (join point), przekroje (cross-cutting concerns)**,
- nauczenie się implementowania prostych aspektów np. logowania i pomiaru czasu w Pythonie,
- zrozumienie, jak AOP umożliwia **separację logiki głównej od przekrojowej**,
- przygotowanie studentów do rozpoznawania miejsc, gdzie AOP przynosi korzyści.

Zakres tematyczny:

- podstawowe pojęcia AOP,
- problemy przekrojowe i separacja odpowiedzialności,
- implementacja aspektów w Pythonie za pomocą dekoratorów i proxy,
- przykłady zastosowań: logowanie, bezpieczeństwo, pomiar czasu, cache.

---

2. Wprowadzenie teoretyczne

2.1 Dlaczego AOP?

W dużych projektach często występują tzw. **zagadnienia przekrojowe** (cross-cutting concerns), które:

- pojawiają się w wielu miejscach kodu,
- nie należą bezpośrednio do logiki biznesowej,
- trudno je utrzymać i modyfikować, jeśli są „rozsiane” po kodzie.

Przykłady:

- logowanie,
- autoryzacja i bezpieczeństwo,
- obsługa wyjątków,
- pomiar czasu i profilowanie,
- cache.

👉 AOP umożliwia **wydzielenie takich zagadnień do osobnych modułów – aspektów**.

2.2 Podstawowe pojęcia AOP

- **Aspekt (Aspect)** – moduł implementujący funkcjonalność przekrojową.

- **Join Point** – konkretne miejsce w programie, w którym można „wstrzyknąć” kod aspektu (np. początek metody, wywołanie funkcji).
- **Advice** – kod wykonywany w join point (np. „przed” lub „po” wywołaniu funkcji).
- **Pointcut** – wzorzec określający, do których join points zastosować aspekt.
- **Weaving** – proces „wplatania” aspektów do programu, zwykle automatyczny.

2.3 AOP vs OOP

- OOP — organizuje kod wokół obiektów i klas.
- AOP — organizuje kod wokół aspektów i przekrojowych funkcjonalności.
- AOP **nie zastępuje** OOP, lecz **uzupełnia** je.

---

3. Tutorial 1 – Prosty aspekt logowania (Python)

Najłatwiejszy sposób implementacji aspektów w Pythonie to **dekoratory**.

3.1 Dekorator logujący

```
def logowanie(funkcja):
    def wrapper(*args, **kwargs):
        print(f"[LOG] Wywołanie {funkcja.__name__} z args={args},
kwargs={kwargs}")
        wynik = funkcja(*args, **kwargs)
        print(f"[LOG] Wynik {funkcja.__name__}: {wynik}")
        return wynik
    return wrapper

@logowanie
def dodaj(a, b):
    return a + b

@logowanie
def podziel(a, b):
    return a / b

print(dodaj(5, 3))
print(podziel(10, 2))
```

👉 Dekorator działa jako **aspekt** logowania — wstrzykuje kod przed i po wywołaniu każdej dekorowanej funkcji.

---

4. Tutorial 2 – Pomiar czasu i łączenie aspektów

4.1 Dekorator mierzący czas

```
import time

def mierz_czas(funkcja):
    def wrapper(*args, **kwargs):
        start = time.time()
        wynik = funkcja(*args, **kwargs)
        end = time.time()
        print(f"[TIMER] {funkcja.__name__} zajęła {end - start:.6f} s")
        return wynik
    return wrapper
```

4.2 Łączenie wielu aspektów

```
@logowanie
@mierz_czas
def suma_naturalnych(n):
    return sum(range(n))

suma_naturalnych(1_000_000)
```

👉 Aspekty można **komponować** – każdy dekorator wprowadza własną logikę przekrojową.

---

5. Tutorial 3 – Proxy obiektowe (zaawansowane)

Dekoratory działają na poziomie funkcji. Można też tworzyć **proxy dla całych obiektów**, które przechwytują wszystkie wywołania metod:

```
class LoggingProxy:
    def __init__(self, obj):
        self._obj = obj

    def __getattr__(self, name):
        attr = getattr(self._obj, name)
        if callable(attr):
            def wrapper(*args, **kwargs):
                print(f"[LOG] Wywołanie metody {name} args={args},
kwargs={kwargs}")
                result = attr(*args, **kwargs)
                print(f"[LOG] Wynik {name}: {result}")
                return result
            return wrapper
        return attr
```

```
# Przykład
class Kalkulator:
    def dodaj(self, a, b): return a+b
    def pomnoz(self, a, b): return a*b

k = Kalkulator()
proxy = LoggingProxy(k)
print(proxy.dodaj(2, 3))
print(proxy.pomnoz(4, 5))
```

👉 Proxy pozwala stosować aspekt do **całych klas** bez modyfikacji ich kodu.

---

6. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Napisz dekorator `autoryzacja`, który:

- sprawdza, czy w zmiennej `uzytkownik` jest odpowiednia rola,
- tylko wtedy pozwala wykonać funkcję,
- w przeciwnym razie wypisuje komunikat o braku uprawnień.

Zadanie 2 (obowiązkowe)

Zaimplementuj dwa aspekty:

- `logowanie`,
- `mierz_czas`.
  Zastosuj je do kilku funkcji, które wykonują obliczenia lub przetwarzają dane.
  Zbadaj, w jakiej kolejności są wykonywane.

Zadanie 3 (dodatkowe)

Napisz **proxy obiektowe**, które dodaje aspekt cache do dowolnego obiektu — zapamiętuje wyniki wywołań metod i zwraca je z pamięci przy powtórnych wywołaniach z tymi samymi argumentami.

---

7. Pytania kontrolne

1. Czym są aspekty i po co się je stosuje?
2. Co to jest join point i advice?
3. Jakie są różnice między AOP a OOP?
4. Jak w Pythonie można zaimplementować AOP?
5. Jakie są potencjalne zastosowania AOP w dużych projektach?

---

8. Checklisty

AOP

- Zaimplementowano co najmniej jeden aspekt jako dekorator.
- Aspekty wstrzykują logikę przed/po wywołaniu funkcji.
- Kod główny nie zawiera logiki przekrojowej.
- Aspekty są wielokrotnego użytku i niezależne.
- (Dla chętnych) Proxy obiektowe działa poprawnie.

---

9. Wzór sprawozdania

Dodatkowo:

- Diagram ilustrujący przepływ wykonania z aspektem.
- Fragmenty kodu funkcji i dekoratorów.
- Krótkie porównanie AOP i wcześniejszych paradygmatów.

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć koncepcję paradygmatu aspektowego,
- umieć identyfikować zagadnienia przekrojowe,
- potrafić implementować proste aspekty w Pythonie,
- stosować dekoratory i proxy do separacji logiki przekrojowej,
- znać podstawowe pojęcia AOP (aspekt, join point, weaving).