Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 11 – Paradygmat zdarzeniowy i reaktywny

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

1. Cel i zakres zajęć

Celem zajęć jest:

- zrozumienie istoty **paradygmatu zdarzeniowego** i **reaktywnego**,
- poznanie sposobu pisania programów sterowanych zdarzeniami,
- nauczenie się implementacji prostych systemów event-driven w Pythonie,
- zapoznanie z pojęciami **emitera zdarzeń**, **obserwatora**, **reakcji** i **strumienia zdarzeń**,
- zaprojektowanie własnego mini-systemu zdarzeniowego lub reaktywnego.

Zakres tematyczny:

- model programowania zdarzeniowego,
- pętle zdarzeń, obsługa zdarzeń, callbacki,
- wzorzec obserwator (observer pattern),
- podstawy programowania reaktywnego,
- przykłady GUI i serwerów opartych o zdarzenia.

---

2. Wprowadzenie teoretyczne

2.1 Programowanie zdarzeniowe

Paradygmat zdarzeniowy opiera się na **reakcji programu na zdarzenia** pochodzące z:

- użytkownika (kliknięcia, wciśnięcia klawiszy),
- środowiska (sygnały, zmiany danych),
- systemu (przybycie wiadomości, zakończenie procesu, itp.).

W takim modelu:

- Program nie wykonuje sekwencji instrukcji sam z siebie.
- Czeka na zdarzenia, a następnie uruchamia **odpowiednie funkcje obsługi (handlers)**.

2.2 Pętla zdarzeń

Większość aplikacji event-driven ma tzw. **pętlę zdarzeń**, która:

1. Nasłuchuje nowych zdarzeń,
2. Przekazuje je do odpowiednich handlerów,
3. Wznawia nasłuchiwanie.

Schemat:

```
inicjalizacja systemu
zarejestruj zdarzenia i handler'y
while True:
    zdarzenie = pobierz_zdarzenie()
    wywołaj_handler(zdarzenie)
```

2.3 Programowanie reaktywne

Programowanie reaktywne to uogólnienie modelu zdarzeniowego, w którym:

- Mamy **strumienie danych (zdarzeń)**,
- Możemy je obserwować, przekształcać, filtrować i łączyć,
- System automatycznie **reaguje na zmiany**, bez jawnego sterowania przepływem.

Stosowane np. w:

- GUI (interfejsy użytkownika),
- Aplikacjach webowych i mobilnych,
- Systemach przetwarzania danych w czasie rzeczywistym.

---

3. Tutorial 1 – Prosty system zdarzeniowy (Python)

3.1 Emitter i handler

```
class EventEmitter:
    def __init__(self):
        self.handlers = {}

    def on(self, event, handler):
        """Rejestruje handler dla danego zdarzenia."""
        if event not in self.handlers:
            self.handlers[event] = []
        self.handlers[event].append(handler)

    def emit(self, event, *args, **kwargs):
        """Wywołuje wszystkie handlery przypisane do zdarzenia."""
        if event in self.handlers:
            for h in self.handlers[event]:
                h(*args, **kwargs)
```

Przykład użycia:

```
def przywitaj(imie):
    print(f"Cześć {imie}!")

def zegnaj(imie):
    print(f"Do zobaczenia, {imie}!")
```

```
emitter = EventEmitter()
emitter.on("powitanie", przywitaj)
emitter.on("pozegnanie", zegnaj)

emitter.emit("powitanie", "Anna")
emitter.emit("pozegnanie", "Anna")
```

3.2 Wzorzec obserwator (Observer Pattern)

```
class Observer:
    def update(self, dane):
        raise NotImplementedError

class Subject:
    def __init__(self):
        self.observers = []

    def attach(self, obs):
        self.observers.append(obs)

    def detach(self, obs):
        self.observers.remove(obs)

    def notify(self, dane):
        for o in self.observers:
            o.update(dane)

# Przykład użycia
class Logger(Observer):
    def update(self, dane):
        print(f"[LOG]: {dane}")

class Alarm(Observer):
    def update(self, dane):
        if dane > 100:
            print("⚠ Alarm! Dane przekroczyły 100")

subject = Subject()
subject.attach(Logger())
subject.attach(Alarm())

subject.notify(50)
subject.notify(120)
```

---

4. Tutorial 2 – Strumienie i reaktywność (prosty przykład)

4.1 Strumień jako generator zdarzeń

```
import time
import random

def strumien_temperatur():
```

```
    while True:
        yield random.randint(0, 150)
        time.sleep(0.5)

def monitoruj_strumien(stream):
    for temp in stream:
        print(f"Temperatura: {temp}")
        if temp > 100:
            print("⚠ Przekroczono próg temperatury!")
monitoruj_strumien(strumien_temperatur())
```

4.2 Reaktywne przekształcenia (filtrowanie, mapowanie)

```
def filtruj(stream, predykat):
    for x in stream:
        if predykat(x):
            yield x

def mapuj(stream, funkcja):
    for x in stream:
        yield funkcja(x)

strumien = strumien_temperatur()
strumien_przefiltrowany = filtruj(strumien, lambda x: x > 50)
strumien_przeksztalcony = mapuj(strumien_przefiltrowany, lambda x: x * 1.8
+ 32)

for temp in strumien_przeksztalcony:
    print(f"Temperatura w °F: {temp}")
```

👉 Widzimy podejście reaktywne — dane „płyną”, a system **reaguje automatycznie** na ich pojawienie się.

---

5. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Zaimplementuj prosty **system zdarzeniowy** (emitter + handlery).
Dodaj co najmniej 3 różne typy zdarzeń i różne handlery.
Symuluj występowanie zdarzeń w pętli czasowej.

Zadanie 2 (obowiązkowe)

Zaimplementuj wzorzec **obserwator**:

- Obiekt „stacja pogodowa” (subject) wysyła dane,
- Obserwatorzy: Logger, Alarm, UI — każdy reaguje inaczej.

Zadanie 3 (dodatkowe)

Napisz prosty **reaktywny filtr strumienia danych**, który:

- generuje losowe liczby,
- filtruje wartości > 500,
- przekształca je np. do postaci tekstowej,
- wypisuje na ekranie w czasie rzeczywistym.

---

6. Pytania kontrolne

1. Na czym polega programowanie zdarzeniowe?
2. Co robi pętla zdarzeń?
3. Na czym polega wzorzec obserwator?
4. Czym różni się podejście reaktywne od imperatywnego?
5. Jakie są potencjalne zastosowania paradygmatu reaktywnego?

---

7. Checklisty

Zdarzeniowe / reaktywne

- Zaimplementowano pętlę zdarzeń lub emitter.
- Zarejestrowano handlery i poprawnie obsłużono zdarzenia.
- Użyto wzorca obserwator w praktyce.
- System działa reaktywnie na dane w czasie rzeczywistym.
- Kod jest modularny i czytelny.

---

8. Wzór sprawozdania

Dodatkowo:

- Schemat działania pętli zdarzeń lub diagram klas obserwatora.
- Fragmenty kodu i przykładowe przebiegi programu.
- Krótka analiza: zalety i ograniczenia podejścia zdarzeniowego w Twoim przykładzie.

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć podstawy paradygmatu zdarzeniowego i reaktywnego,
- umieć pisać programy reagujące na zdarzenia w czasie rzeczywistym,
- znać wzorzec obserwator i jego zastosowanie,
- rozumieć różnicę między podejściem reaktywnym a imperatywnym,
- potrafić projektować proste systemy event-driven.