Paradygmaty programowania

Paradygmat - zbiór mechanizmów, jakich programista używa, pisząc program oraz jak ów program jest następnie wykonywany przez komputer. Można zatem powiedzieć, że paradygmat programowania to ogół oczekiwań programisty wobec języka programowania i komputera, na którym będzie działał program.

Programowanie obiektowe

W programowaniu obiektowym program to zbiór porozumiewających się ze sobą obiektów, czyli jednostek zawierających określone dane i umiejących wykonywać na nich określone operacje. Najważniejsze są tu dwie cechy:

- powiązanie danych (czyli stanu) z operacjami na nich (czyli poleceniami) w całość, stanowiącą odrębną jednostkę — obiekt;
- mechanizm dziedziczenia, czyli możliwość definiowania nowych, bardziej złożonych obiektów, na bazie obiektów już istniejących.

Z tych dwóch cech bierze się zapewne wielki sukces paradygmatu obiektowego. Umożliwia on bowiem modelowanie zjawisk rzeczywistego świata w uporządkowany, hierarchiczny sposób — od idei do szczegółów technicznych. Wśród najważniejszych języków należy wymienić C++ (chociaż nie jest to język czysto obiektowy) i Javę

Przykład

```python
# Definicja klasy "Prostokat"
class Prostokat:
    def __init__(self, dlugosc, szerokosc):
        self.dlugosc = dlugosc
        self.szerokosc = szerokosc

    def oblicz_pole(self):
        return self.dlugosc * self.szerokosc

    def oblicz_obwod(self):
        return 2 * (self.dlugosc + self.szerokosc)


# Główna część programu
if __name__ == "__main__":
    # Tworzenie obiektu klasy "Prostokat"
    moj_prostokat = Prostokat(5, 3)

    # Obliczanie pola i obwodu prostokąta
    pole = moj_prostokat.oblicz_pole()
    obwod = moj_prostokat.oblicz_obwod()

    # Wyświetlanie wyników
    print(f"Pole prostokąta wynosi: {pole}")
    print(f"Obwód prostokąta wynosi: {obwod}")
```

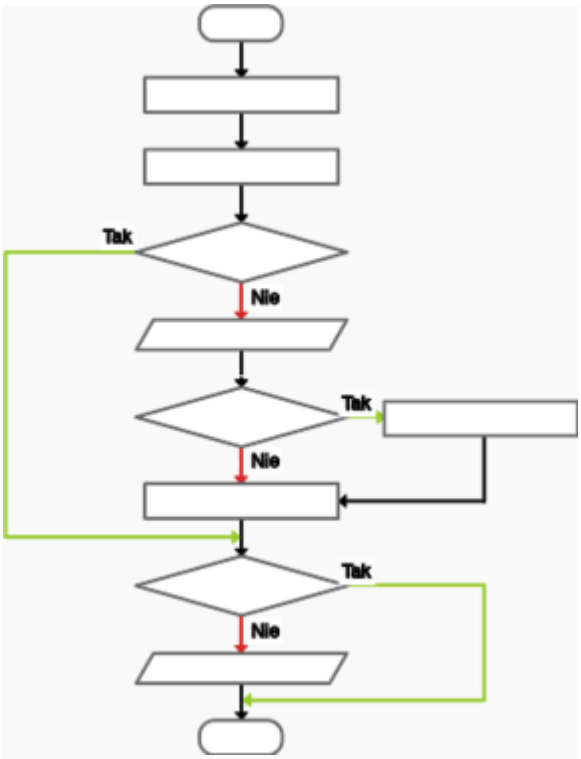
- Tworzymy klasę Prostokat z konstruktorem __init__, który inicjuje długość i szerokość prostokąta.
- Klasa ma dwie metody: oblicz_pole, która oblicza pole prostokąta, i oblicz_obwod, która oblicza obwód prostokąta.
- Następnie tworzymy obiekt moj_prostokat na podstawie klasy Prostokat.
- Wywołujemy metody obliczające pole i obwód prostokąta na tym obiekcie i wyświetlamy wyniki.
- Ten przykład ilustruje paradygmat programowania obiektowego, gdzie dane i metody są zorganizowane wokół obiektów, co pozwala na bardziej czytelny i modułowy kod.

Programowanie strukturalne

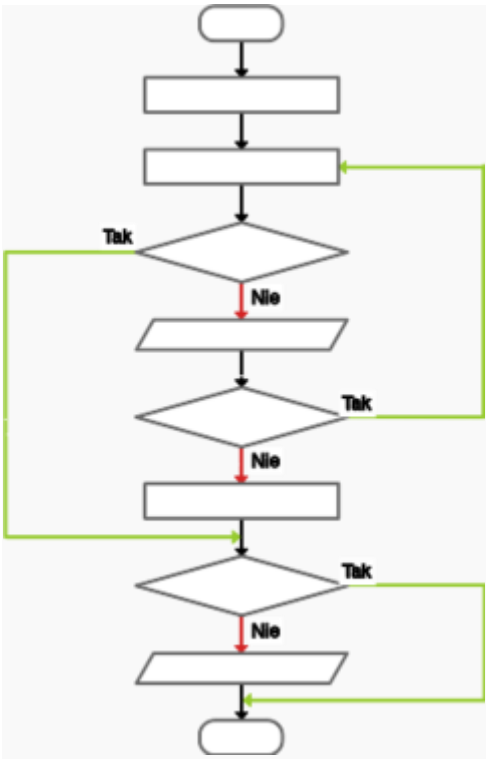
Chodzi w nim o tworzenie programów z kilku dobrze zdefiniowanych konstrukcji takich jak instrukcja warunkowa if-else i pętla while. Powinno to sprzyjać pisaniu programów przejrzystych, łatwych w rozumieniu i utrzymaniu.

Ściślej, Dijkstra proponował użycie tylko trzech rodzajów struktur sterujących:

- Sekwencja (lub konkatenacja) — czyli po prostu wykonanie instrukcji w określonej kolejności. W wielu językach rolę „operatora konkatenacji instrukcji" spełnia niepozorny średnik...
- Wybór — czyli wykonanie jednej z kilku instrukcji zależnie od stanu programu. Przykładem jest if-then-else i switch/case.
- Iteracja, czyli powtarzanie instrukcji tak długo, jak długo spełniony (lub niespełniony) jest dany warunek. Chodzi oczywiście o pętle, np. while, repeat-until, for itp.



Programowanie strukturalne



Programowanie niestrukturalne

Powyższe struktury stosuje się do „małych" jednostek programu, złożonych z elementarnych instrukcji, czyli np. podstawień, wywołań procedur lub instrukcji wejścia-wyjścia. „Duże" jednostki powinny być — wedle zasad programowania strukturalnego — rozbite na mniejsze (funkcje, procedury, bloki itp.), **tak by można było zrozumieć poszczególne fragmenty bez rozumienia całości.**

Przykład

```python
# Definicja funkcji do obliczania sumy dwóch liczb
def dodawanie(a, b):
    return a + b

# Definicja funkcji do obliczania różnicy dwóch liczb
def odejmowanie(a, b):
    return a - b

# Definicja funkcji do obliczania iloczynu dwóch liczb
def mnozenie(a, b):
    return a * b

# Definicja funkcji do obliczania ilorazu dwóch liczb
def dzielenie(a, b):
    if b != 0:
        return a / b
    else:
        return "Nie można dzielić przez zero."

# Główna część programu
if __name__ == "__main__":
    # Wprowadź dane
    liczba1 = float(input("Podaj pierwszą liczbę: "))
    liczba2 = float(input("Podaj drugą liczbę: "))

    # Wybierz operację
    print("Wybierz operację:")
    print("1. Dodawanie")
    print("2. Odejmowanie")
    print("3. Mnożenie")
    print("4. Dzielenie")

    wybor = input("Podaj numer operacji: ")

    if wybor == "1":
        wynik = dodawanie(liczba1, liczba2)
    elif wybor == "2":
        wynik = odejmowanie(liczba1, liczba2)
    elif wybor == "3":
        wynik = mnozenie(liczba1, liczba2)
    elif wybor == "4":
        wynik = dzielenie(liczba1, liczba2)
    else:
        wynik = "Nieprawidłowy wybór operacji."

    print("Wynik: ", wynik)
```

Ten przykład kodu demonstruje podejście strukturalne, w którym program jest zorganizowany wokół funkcji, a sterowanie przekazywane jest na podstawie wyboru użytkownika. Funkcje są oddzielone i wykonują konkretne zadania, co sprawia, że kod jest bardziej czytelny i modułowy.