Sprawozdanie z laboratoriów nr 2

Dawid Ochman

Informatyka

III rok, I stopień, grupa laboratoryjna nr 3

Semestr Zimowy 2025/2026

1. Implementacja rozwiązania

Interfejsy: IVector, IPolar2D

Klasy bazowe: Vector2D, Vector3D, Polar2D

Wzorce projektowe:

├── Adapter: Polar2DAdapter

├── Decorator: Vector3DDecorator

└── Inheritance: Vector3DInheritance

2. ADAPTER PATTERN - Polar2DAdapter

Klasa adaptująca Vector2D do interfejsu IPolar2D, umożliwiając reprezentację wektora kartezjańskiego jako biegunowego.

Zalety

- Separacja odpowiedzialności: Vector2D nie musi znać współrzędnych biegunowych
- Elastyczność: Dynamiczna zmiana reprezentacji bez modyfikacji oryginału
- Open/Closed Principle: Rozszerzenie funkcjonalności bez zmiany kodu

Wady

- Narzut wydajnościowy: Dodatkowa warstwa wywołań przez wskaźnik
- Zarządzanie pamięcią: Konieczność dbania o cykl życia adaptowanego obiektu
- Złożoność: Dodatkowa klasa w projekcie

Zastosowanie w Projekcie : Użyto do wyświetlania wektorów 2D w układzie biegunowym bez modyfikacji klasy Vector2D.

3. DECORATOR PATTERN - Vector3DDecorator

Dodaje trzeci wymiar (Z) do dowolnego IVector poprzez kompozycję.

Zalety

- Dynamiczne rozszerzanie: Dodanie wymiaru Z w runtime
- Kompozycja > dziedziczenie: Bardziej elastyczne rozwiązanie
- Uniwersalność: Działa z dowolnym IVector

Wady

- Narzut pamięciowy: Każdy dekorator = nowy obiekt + wskaźnik
- Trudniejsze debugowanie: Wiele warstw abstrakcji

- Problemy z castowaniem: dynamic_cast może zawieść na zagnieżdżonych dekoratorach

Zastosowanie w Projekcie : Umożliwia traktowanie wektora 2D jako 3D bez tworzenia nowej instancji - przydatne gdy z=0 jest wartością domyślną.

4. **INHERITANCE PATTERN - Vector3DInheritance**

Rozszerzenie Vector2D poprzez klasyczne dziedziczenie, dodające składową Z.

Zalety

- Wydajność: Brak wskaźników - najszybsze rozwiązanie
- Prostota kodu: Intuicyjne i czytelne
- Bezpośredni dostęp: Do pól x, y klasy bazowej (protected)
- Optymalizacja: Kompilator może inline'ować metody

Wady

- Sztywność: Niemożliwa zmiana hierarchii w runtime
- Silne powiązanie: Zmiana Vector2D wpływa na klasę pochodną
- Fragile Base Class: Modyfikacja bazowej klasy może złamać dziedziczącą
- Brak elastyczności: Nie można "przekształcić" 2D→3D dynamicznie

Zastosowanie w Projekcie : Użyto jako najbardziej naturalnego sposobu reprezentacji wektora 3D, który "jest" wektorem 2D z dodatkowym wymiarem.

5. ILOCZYN WEKTOROWY - METODA MACIERZOWA

Zalety

- Matematyczna poprawność: Bezpośrednie przełożenie wzoru na wyznacznik
- Uniwersalność: Obsługuje wektory 2D (z=0) i 3D
- Czytelność: Kod odzwierciedla wzór matematyczny

Wady

- 6 mnożeń: Więcej operacji niż potencjalne optymalizacje
- Brak SIMD: Trudniejsza wektoryzacja

6. Podsumowanie

Projekt demonstruje trzy fundamentalne podejścia rozszerzania funkcjonalności w C++:

1. Adapter - najlepszy do konwersji interfejsów
2. Decorator - optymalny dla dynamicznych rozszerzeń
3. Inheritance - najprostszy i najszybszy dla stabilnych hierarchii

Wybór wzorca zależy od:

- Wymagań wydajnościowych
- Potrzeby elastyczności
- Złożoności systemu
- Etapu życia aplikacji (design vs runtime)

Każde podejście ma swoje miejsce w profesjonalnym programowaniu - klucz to świadomy wybór odpowiedniego narzędzia do problemu.