Sprawozdanie

Analiza podejść użytych w projekcie „VectorMathLab”

W przedstawionym projekcie zaprojektowano zestaw klas i interfejsów służących do reprezentowania i przetwarzania wektorów w różnych układach współrzędnych (kartezjańskim i biegunowym). Projekt łączy różne techniki programowania obiektowego, takie jak interfejsy, dziedziczenie oraz wzorce projektowe **Adapter** i **Dekorator**. Celem było uzyskanie elastycznej, rozszerzalnej struktury umożliwiającej łatwe dodawanie nowych typów wektorów bez ingerencji w istniejący kod.

Interfejsy

Interfejsy *IVector i IPolar2D* określają wspólny zestaw metod, które muszą być zaimplementowane przez konkretne klasy wektorów.
Zaletą tego podejścia jest **spójność i elastyczność** — możemy dowolnie wymieniać implementacje bez zmiany reszty kodu. Interfejsy ułatwiają też testowanie i pozwalają na późniejsze dodawanie nowych typów wektorów.
Minusem jest natomiast konieczność powtarzania implementacji podobnych metod w wielu klasach, co w pewnym stopniu zwiększa ilość kodu.

Dziedziczenie

Klasy takie jak *Vector2D, Vector3DInheritance czy 2DPolarInheritance* korzystają z klasycznego dziedziczenia. Dzięki temu można łatwo rozbudowywać istniejące klasy, np. dodać trzeci wymiar lub funkcje trygonometryczne bez przepisywania wcześniejszej logiki.
Główna zaleta to **czytelność i prostota** – łatwo zrozumieć, co rozszerza co.
Wadą jest jednak **sztywność hierarchii** – jeśli w przyszłości pojawi się potrzeba zmiany struktury lub wprowadzenia nowych typów wektorów, może być trudno uniknąć konfliktów metod i nadpisywania funkcji.

Wzorzec Adapter (*Polar2DAdapter*)

Adapter umożliwia wykorzystanie klasy *Vector2D* w inny sposób – w tym przypadku pozwala przeliczyć współrzędne na układ biegunowy bez zmiany samej klasy.
Jest to bardzo **czyste i elastyczne** rozwiązanie – logika konwersji jest odseparowana od logiki obliczeniowej.
Minusem może być zwiększona liczba klas pośrednich, co może utrudnić orientację w projekcie przy pierwszym kontakcie z kodem.

Wzorzec Dekorator (*Vector3DDecorator*)

Dekorator to ciekawy sposób na **dynamiczne rozszerzanie funkcjonalności** – tutaj pozwala „dodać” wymiar *z* do dowolnego wektora, bez modyfikowania jego kodu źródłowego.
Zaletą jest ogromna elastyczność – można tworzyć złożone obiekty poprzez nakładanie dekoratorów.
Wadą jest natomiast **zwiększona złożoność** kodu – każdy dekorator wprowadza dodatkowy poziom pośrednictwa, co utrudnia śledzenie przepływu danych.

Podsumowanie

Projekt dobrze ilustruje różne podejścia do rozbudowy funkcjonalności w programowaniu obiektowym.

- **Interfejsy** – zapewniają spójność i możliwość wymiany implementacji.
- **Dziedziczenie** – proste, ale mało elastyczne przy większej liczbie rozszerzeń.
- **Adapter** – ułatwia współpracę między różnymi modelami danych.
- **Dekorator** – pozwala dynamicznie rozszerzać zachowanie obiektów.

Każde z tych podejść ma swoje zastosowanie – wybór zależy od tego, czy priorytetem jest prostota, elastyczność czy łatwość rozbudowy. W tym projekcie połączenie wszystkich czterech koncepcji dało elastyczny, a jednocześnie przejrzysty model obsługi wektorów w różnych układach współrzędnych.