Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki
i Matematyki

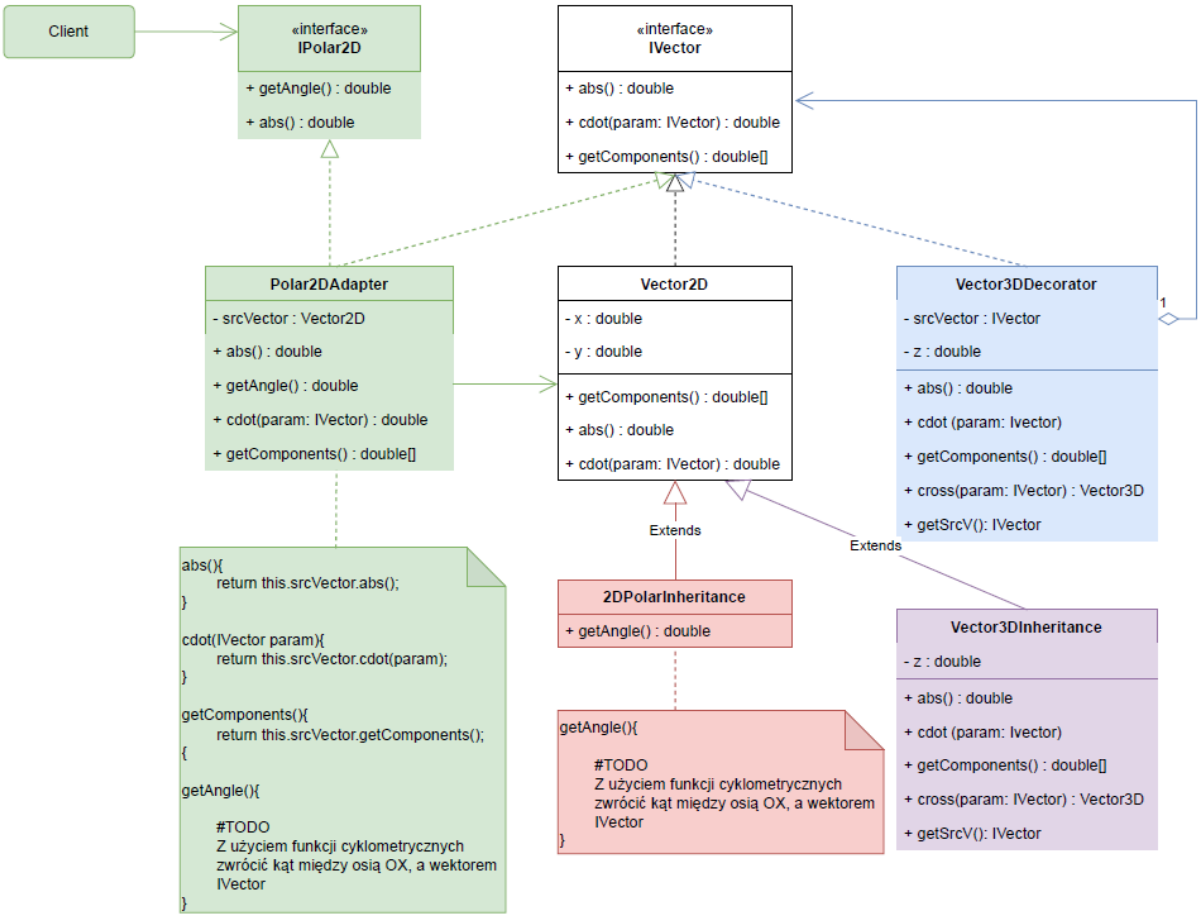
TECHNOLOGIE OBIEKTOWE
Laboratorium 2:
Adapter oraz dekorator w przykładach

przedmiot: ..... **Technologie Obiektowe**
forma/stopień/rok/semestr studiów:... **stacjonarne/I/III/5**
kierunek/specjalność: ..... **informatyka/brak specjalności**
semestr i rok akademicki: ..... **zimowy 2025/2026**
imię i nazwisko studenta: ..... **Michał Brzeziński**
nr albumu studenta: **151408**

Data wydania ćwiczenia:	Data oddania ćwiczenia:	Prowadzący:
16.10.2025 r.	**22.10.2025 r.**	**mgr inż. Piotr Szuster**

Treść zadania

Zaimplementować klasy przestawione na diagramie UML. W funkcji main() utworzyć trzy przykładowe wektory o wybranych współrzędnych. Dla każdego z nich wyświetlić na ekranie ich współrzędne w układach kartezjańskim oraz biegunowym. Wyświetlić wyniki iloczynu skalarnego oraz wektorowego (w formie współrzędnych kartezjańskich) dla wszystkich możliwych kombinacji.



Rys. 1 Diagram UML aplikacji

Implementacja aplikacji została zrealizowana w języku JAVA

Zestawienie wad i zalet obu podejść

DZIEDZICZENIE

ZALETY:

- Bardzo proste do zrozumienia i zaimplementowania dla prostych hierarchii.
- Relacja 'Vector3D' JEST 'Vector2D' (z dodatkami) jest intuicyjna.
- Brak narzutu na delegowanie wywołań; wywołania metod są bezpośrednie.

WADY:

- Hierarchia jest ustalana na etapie kompilacji. Nie można jej zmienić w trakcie działania programu.
- Obiekt nie może być JEDNOCZEŚNIE Polar2D i Vector3D. (Musielibyśmy stworzyć czwartą klasę 'PolarVector3DInheritance', co prowadzi do 'eksplozji klas').
- Łamanie hermetyzacji: Klasy pochodne są mocno powiązane z implementacją klasy bazowej.

WZORCE PROJEKTOWE (Adapter i Dekorator)

ZALETY:

- Elastyczność: Możemy dynamicznie (w trakcie działania programu) 'dekorować' obiekty, dodając funkcje.
- Lepsza kompozycja: Możemy łączyć funkcje. Obiekt może być jednocześnie 'polarny' (przez Adapter) i '3D' (przez Dekorator).
- Zgodność z 'Open/Closed Principle': Możemy dodawać nowe dekoratory (np. Vector4DDecorator) bez modyfikowania istniejącego kodu.
- Zgodność z 'Single Responsibility Principle': Każda klasa robi jedną rzecz (Vector2D przechowuje dane, Adapter adaptuje, Dekorator dodaje wymiar).

WADY:

- Większa złożoność, ponieważ wymaga większej liczby klas, co może być trudniejsze do zrozumienia.
- Narzut wydajnościowy: (Niewielki) koszt 'delegowania' wywołać (np. Vector3DDecorator wywołuje srcVector.getComponents()).