|  |  |  |
| --- | --- | --- |
| Aleksandra Duda | Gr. 31 lab 02 | Techniki Obiektowe |
| Nr indeksu: 130510 | Laboratorium nr 2 | 13.10.20 r. |

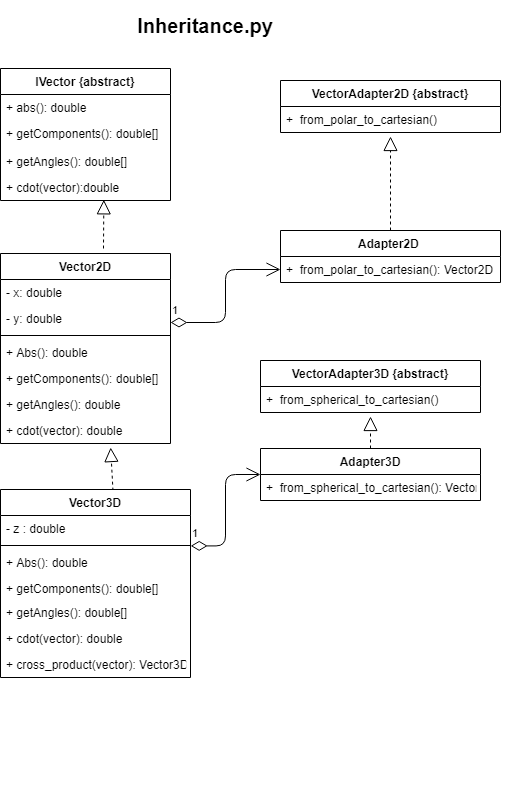
**Sprawozdanie z labolatorium nr 2**

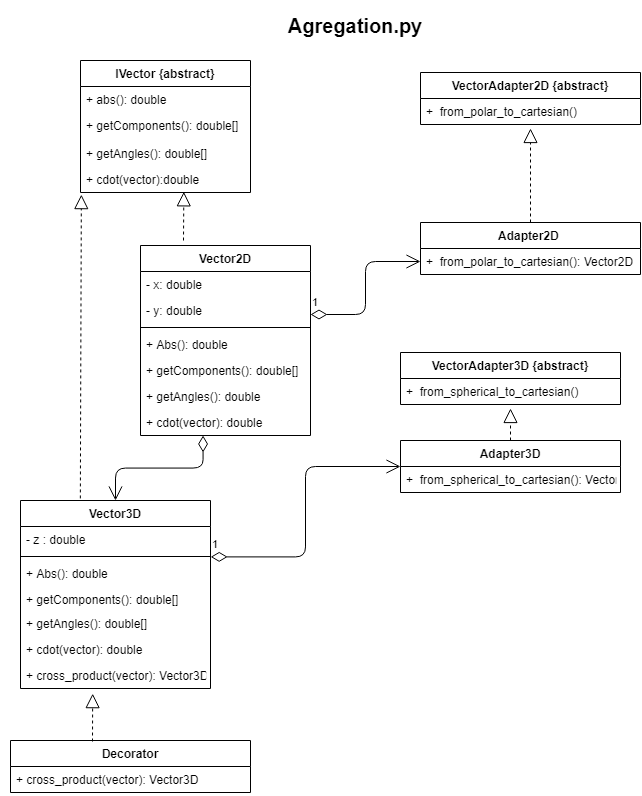
1. **Cel ćwiczenia**

Implementacja interfejsu IVector oraz klass Vector2d, Vector3d wraz z metodami.

Należało również umożliwić obliczanie iloczynu wektorowego oraz tworzenie wektorów 2D oraz 3D na podstawie współrzędnych biegunowych. W projekcie należało uwzględnić takie wzorce jak dekoder i adapter. Należało wykorzystać również dziedziczenie i agregację.

1. **Moele UML**

****



1. **Przebieg ćwiczenia**

**Plik „Inheritance.py” – wersja z dziedziczeniem**

* Z racji tego, że tworzyłam projekt w Pythonie i w tym języku raczej interfejsów się nie praktykuje to zastąpiłam go klasą abstrakcyjną IVector.



Deklarujemy metody , które będą nam potrzebne do działań na wektorach, ale ich tutaj nie implementujemy.

* Klasa Vector2D ma implementować IVector – w jednym przypadku w pliku **Inheritance.py** robię to przy pomocy dziedziczenia:



W klasie Vector2d definiuję funkcje działające na wektorach dwuwymiarowych.

**Abs()**  zwraca długość wektora, **getComponents()** zwraca składowe wektora, **getAngles()** zwraca kąt pomiędzy wektorem, a osią OX, **cdot()**  oblicza iloczyn skalarny dwóch wektorów.

* Następnie stworzyłam klasę Vector3D, która obsługuje wektory trójwymiarowe. Klasa Vector3D dziedziczy po klasie Vector2D i przeciąża metody wcześniej zdefiniowane w Vectorze2D.



Przy pomocy dziedziczenia pozyskuję dostęp do metod **Vector2D** dla klasy **Vector3D**. **Super()** służy do wywoływania metody z klasy bazowej w klasie pochodnej – w tym przypadku wywołuję funkcję **getComponents()** z klasy **Vector2D** w klasie **Vector3D** , aby mieć współrzędne x i y.

Klasa **Vector3D**  ma te same metody co klasa **Vector2D** , jednak w klasie **Vector3D**  musieliśmy te metody poprzeciążać i dodać jeszcze jedną metodę **cross\_product(),**  czyli iloczyn wektorowy dwóch wektorów trójwymiarowych.

* Zaimplementowałam również adaptery dla wektorów dwuwymiarowych, jak i trójwymiarowych.

Dla wektora dwuwymiarowego:



W tym adapterze przechodzimy od układu biegunowego do kartezjańskiego.

Odwołujemy się do klasy abstrakcyjnej **VectorAdapter2D** (w Pythonie raczej nie korzysta się z interfejsów) , który ma zadeklarowaną metodę konwersji ze współrzędnych polarnych na kartezjańskie. Klasa **Adapter2D** implementuje ten interfejs, metoda **from\_polar\_to\_cartesian()** przekształca współrzędne i oddelegowuje do tworzenia wektora 2d w konstruktorze klasy **Vector2D**.

Podobnie zrobiłam dla wektorów 3D:



**Plik „Agregation.py” – wersja z agregacją**

* Jedną z różnic między programami **Inheritance.py** , a **Agregation.py**  jest sposób pozyskiwania metod i atrybutów klasy **Vector3D**  od klasy **Vector2D.**

Klasa **Vector3D** dziedziczy po klasie abstrakcyjnej **IVector**, zamiast po **Vector2D.** W konstruktorze klasy **Vector3D** jest tworzony obiekt klasy **Vector2D**, co nazywamy właśnie agregacją – **Vector2D** zawiera się w **Vector3D.**

****

* W tym przypadku mamy również do czynienia z **Dekoratorem**, który reprezentuje klasa **Decorator**  w którym zawiera się metoda do liczenia iloczynu wektorowego.

****

Metoda **cross\_product**() oblicza wartości wektora wynikowego (rezultatu iloczynu wektorowego) po czym „oddelegowuje” tworzenie nowego wektora do konstruktora klasy **Vector3D**  gdzie będzie tworzony nowy wektor.

1. **Wnioski**

* Klasę **IVector** możemy implementować na dwa sposoby – poprzez dziedziczenie lub agregację.

Jeśli wybraliśmy dziedziczenie to może wystąpić problem modyfikatorów dostępu.

Są na to dwa rozwiązania:

- oznaczenie atrybutów klasy jako private, następnie stworzenie metody, która pobiera i zwraca te atrybuty klasy. (zastosowane w moich programach)

- zmiana modyfikatorów dostępu na protected tak aby klasa pochodna mogła swobodnie korzystać z atrybutów klasy bazowej.

Osobiście uważam, że drugi sposób jest szybszy, ale mniej bezpieczny.

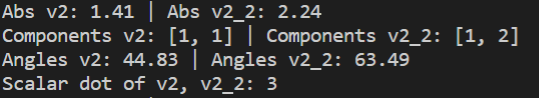
* Dzięki temu, że korzystaliśmy z dziedziczenia (Vector3D dziedziczył z Vector2D) wystarczyło dodać tylko współrzędną z do wektora 3d – nie musieliśmy tworzyć dodatkowo współrzędnych x i y.
* Podoba mi się stosowanie adapterów, bo jest przede wszystkim widoczne rozgraniczenie pomiędzy odpowiedzialnościami w programie – patrząc na **Adapter3D** wiem, że ta klasa ma za zadanie konwertować współrzędne sferyczne do kartezjańskich. Zapobiega to niepotrzebnemu zamieszaniu.
* Aby mieć pewność, że obliczenia zostały poprawnie wykonane to kazdy wynik porównywałam z <https://www.wolframalpha.com/> , <https://www.symbolab.com/>, czy <https://keisan.casio.com/>. Wszystkie wyniki jakie sprawdziłam zgadzały się z tymi na stronie, co oznacza, że alorytmy obliczania długości wektora, liczenie kątów, iloczyn skalarny i wektorowy, czy konwersje na współrzędne kartezjańskie są poprawne.

**Przykład dla klasy Vector2D**

Dlawartości:

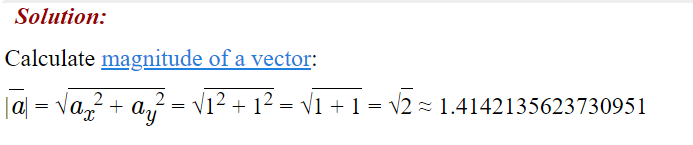
* **v2 = (1,1)** oraz **v2\_2 = (1,2)**
* **R = 3** oraz **angle =30**

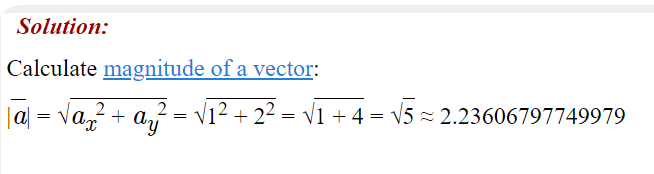
Wyniki z mojego programu:

Wyniki z kalkulatorów online:

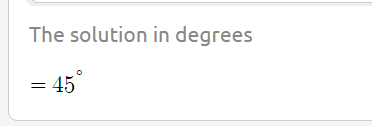
* Długość wektora

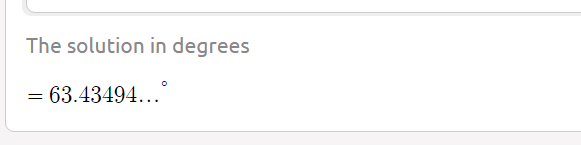




Takie same wyniki z tym, że u mnie są one zaokrąglone.

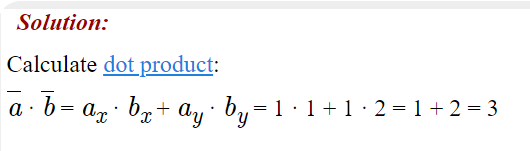
* Kąt





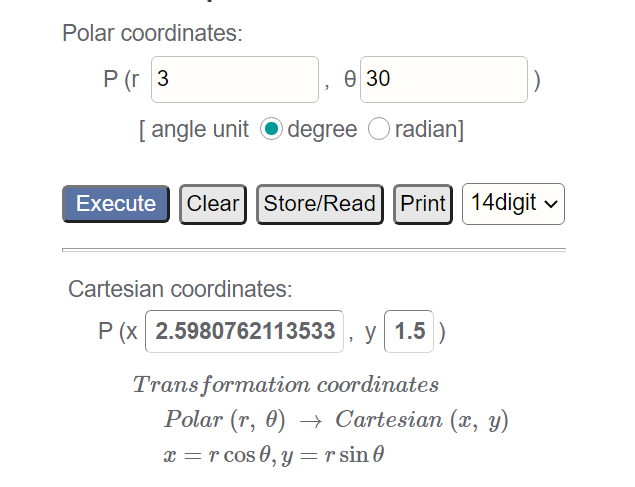
Drobne róznice w wynikach wynikają z użytych przeze mnie innych przybliżeń w moim programie.

* Iloczyn skalarny



Takie same wyniki.

* Ze współrzędnych biegunowych na kartezjańskie



Takie same wyniki z tym, że u mnie są zaokrąglone.