Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

Zadanie 4

Zaawansowane techniki programowania

Prowadzący: dr inż. Jerzy Jaworowski

II stopień - rok 1, semestr 1

wykonanie:

*Piotr Adam Tomaszewski*

*Nr albumu: 104896*

# Wstęp

Celem zadania było obliczenie objętości graniastosłupa prostego o podstawie P oraz wysokości h. Na samym początku należało utworzyć połączenie z bazą danych SQL z wykorzystaniem usługi JNDI. Następnie należało pobrać wartości wierzchołków w przestrzeni 3D z bazy danych i zapisać je w programie. Pole graniastosłupa należało wyznaczyć z wykorzystaniem otoczki wypukłej rzutu punktów x i y na płaszczyznę. Wysokość graniastosłupa należało wyznaczyć ze średniej wartości współrzędnych z.

# Organizacja plików

## IGameRemote.java

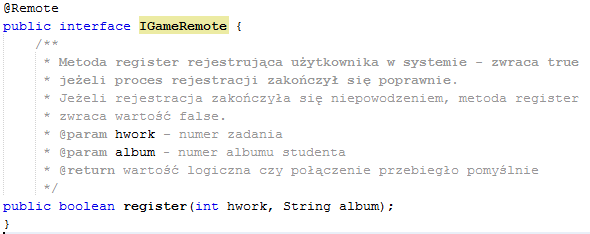
Interfejs ***IGameRemote*** posiadający deklarację metody, która pozwala na zarejestrowanie użytkownika w systemie.

Metoda ***register*** rejestrująca użytkownika w systemie - zwraca jeżeli proces rejestracji zakończył się poprawnie. Jeżeli rejestracja zakończyła się niepowodzeniem, metoda register zwraca wartość .

@param **hwork** – numer zadania

@param **album** – numer albumu studenta

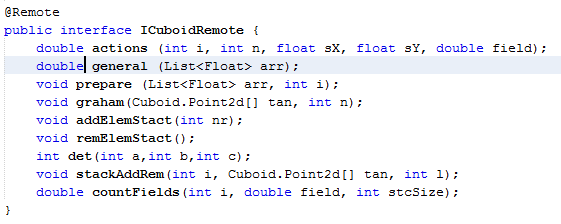
@return wartość logiczna czy połączenie przebiegło pomyślnie



Rysunek Implementacja interfejsu IGameRemote

## ICuboidRemote

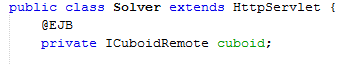
Interfejs ***ICuboidRemote*** zawierający deklaracje metod wykorzystywanych w klasie Cuboid niezbędny do użycia bean'a przekazującego listę punktów z servletu do klasy Cuboid.



Rysunek Implementacja interfejsu ICuboidRemote

## Solver.java

Klasa ***Solver*** będąca servletem pozwalającym na połączenie między aplikacją a bazą danych.



Rysunek Implementacja servletu Solver

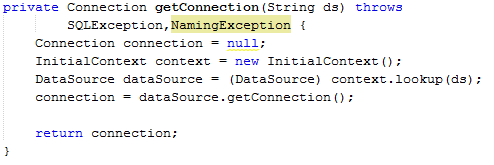
Metoda ***getConnection*** łącząca się z bazą danych za pomocą nazwy JNDI na serwerze Glassfish.

@param **ds** - nazwa JNDI (parametr poprany metodą GET)

@return - połączenie z bazą

@throws **SQLException** - zapewnia informacje o błędzie dostępu do bazy danych

@throws **NamingException** - zapewnia informacje o błędzie gdy nazwa JNDI jest niepoprawna

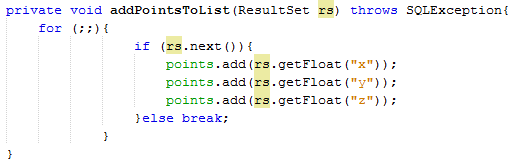


Rysunek Implementacja metody getConnection

Metoda ***addPointsToList*** pobierająca wartości punktów z bazy i zapisująca je do listy.

@param **rs** - zbiór rekordów zwróconych po wykonaniu zapytania w bazie danych

@throws **SQLException** - zapewnia informacje o błędzie dostępu do bazy danych



Rysunek Implementacja metody addPointsToList

Metoda ***getPoints*** w której pobierane są wyniki zapytania w bazie danych i przekazywane są do zapisania na liście.

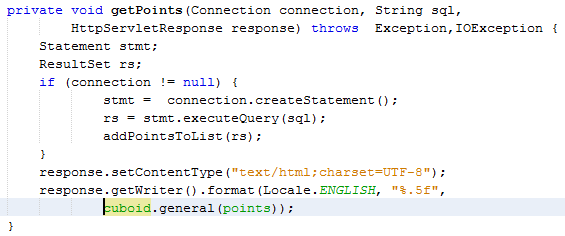
@param **connection** - referencja do otwartego połączenia z bazą danych

@param **sql** - treść zapytania niezbędna do pobrania rezultatu

@param **response** - odpowiedź servletu

@throws **Exception** -przechwycenie zdarzenia, które może zakłócić poprawne działanie programu

@throws **IOException** – w przypadku wystąpienia błędu wejścia/wyjścia



Rysunek Implementacja metody getPoints

Metoda ***doGet***, dzięki której po przekazaniu parametru w żądaniu URL, następuje rejestracja użytkownika i jeśli przebiegnie pomyślnie wykonywane są metody określające połączenie z bazą i pobranie danych z bazy.

@param **request** - żądanie servletu

@param **response** - odpowiedź servletu

@throws **ServletException** – w przypadku wystąpienia błędu związanego z servletem

@throws **IOException** – w przypadku wystąpienia błędu wejścia/wyjścia



Rysunek Implementacja metody doGet

## Cuboid.java

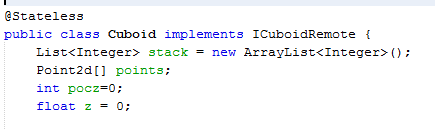
Klasa ***Cuboid*** implementująca metody interfejsu ICuboidRemote służące do obliczenia objętości graniastosłupa prostego mając jego wierzchołki w przestrzeni 3D przy użyciu m.in. algorytmu Grahama. Posiada pola:

**stack** - reprezentujący stos zaimplementowany jako lista;

**points** - tablica zawierająca zbiór punktów w przestrzeni 2D wykorzystywana w algorytmie Grahama.

**pocz** - wartownik wskazujący ilość elementów na stosie

**z** - suma wartości indeksów z w przestrzeni 3D

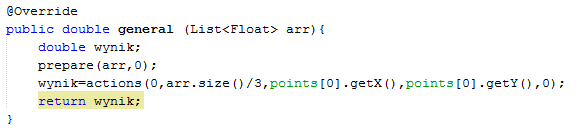


Rysunek Implementacja klasy Cuboid

Główna metoda ***general*** wywoływana przez bean z serwletu określająca wykonywanie kolejnych metod w klasie.

@param **arr** - lista wszystkich pobranych punktów z bazy danych

@return - objętość graniastosłupa

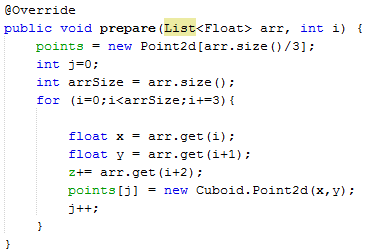


Rysunek Implementacja metody general

Metoda ***prepare*** przyporządkowująca wartości punktów z listy do tablicy obiektów w przestrzeni 2D. Metoda ma na celu odseparowanie ciągu liczb na odpowiadające im współrzędne X i Y. Współrzędna Z jest od razu sumowana aby wyznaczyć jej średnią.

@param **arr** - lista wszystkich pobranych punktów z bazy danych

@param **i** - zmienna pomocnicza do iteracji po elementach



Rysunek Implementacja metody prepare

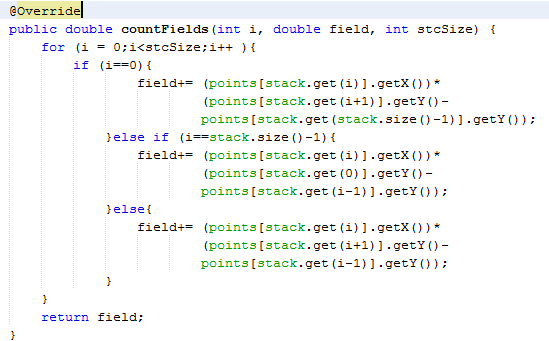
Metoda ***countFields*** pozwalająca obliczyć pole figury zrzutowanej na przestrzeń dwuwymiarową za pomocą wzoru do analitycznego obliczania pól.

@param **i** - zmienna pomocnicza wykorzystywana do iteracji

@param **field** - zmienna pomocnicza przechowująca wartość pola

@param **stcSize** - rozmiar stosu

@return - ostateczna wartość pola



Rysunek Implementacja metody countFields

Metoda ***actions*** wykonująca kolejne kroki pozwalające wyznaczyć pole graniastosłupa przy wykorzystaniu otoczki wypukłej. Po załadowaniu elementów do tablic następuje sortowanie ich w tablicy. Potem wywoływane są metody odpowiedzialne za algorytm Grahama i następuje wypisanie objętości graniastosłupa.

@param **i** - zmienna pomocnicza wykorzystywana do iteracji

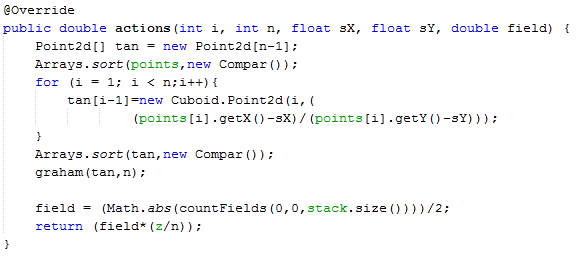
@param **n** - ilość wierzchołków graniastosłupa

@param **sX** - współrzędna x punktu początkowego

@param **sY** - współrzędna y punktu początkowego

@param **field** - zmienna przechowująca wartość pola

@return - objętość graniastosłupa



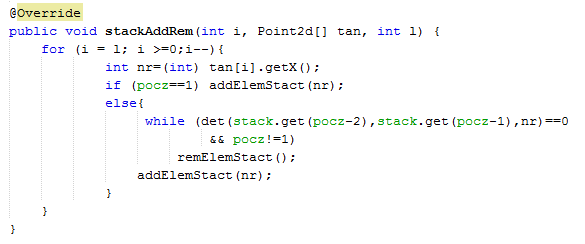
Rysunek Implementacja metody actions

Metoda ***stackAddRem*** algorytmu Grahama pobierająca każdorazowo wartość iteracyjną z tablicy (getX) i dodająca do stosu. W zależności od wyniku wyznacznika wartości są usuwane ze stosu (usuwanie punktów, które spowodowały by zbudowanie otoczki wklęsłej).

@param **i** - zmienna pomocnicza wykorzystywana do iteracji

@param **tan** - tangens nachylenia punktu do osi X

@param **l** - rozmiar tablicy tangens

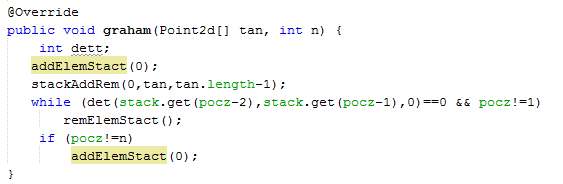


Rysunek Implementacja metody stackAddRem

Metoda ***graham*** implementująca w głównej części algorytm Grahama określający kierunki w których przechodzimy do kolejnego punktu tak, by zbudować otoczkę wypukłą. Do pomocy wykorzystywany jest stos.

@param **tan** - tangens nachylenia punktu do osi X

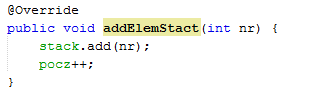
@param **n** - ilość wierzchołków graniastosłupa



Rysunek Implementacja metody graham

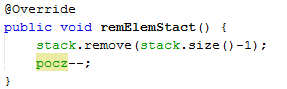
Metoda ***addElemStact*** dodająca do stosu wartość i inkrementująca ilość znajdujących się na nim elementów.

@param **nr** - wartość do dodania na stos



Rysunek Implementacja metody addElemStack

Metoda ***remElemStact*** usuwająca wartości ze stosu i dekrementująca ilość znajdujących się na nim elementów.



Rysunek Implementacja metody remElemStack

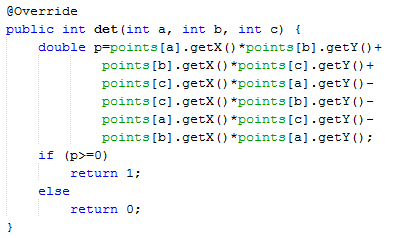
Metoda ***det*** zwracająca wartość wyznacznika ze współrzędnych punktów zawartych w tablicy współrzędnych graniastosłupa

@param **a** - indeks trzeciej współrzędnej

@param **b** - indeks drugiej współrzędnej

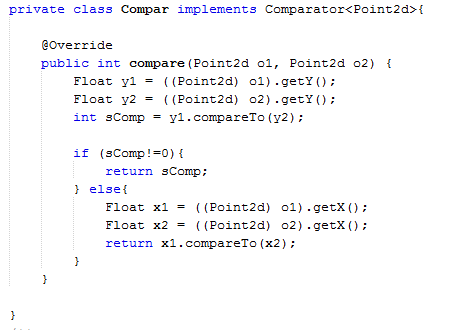
@param **c** - indeks pierwszej współrzędnej

@return określenie czy wyznacznik jest dodatni czy ujemny



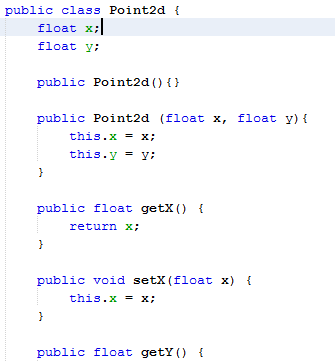
Rysunek Implementacja metody det

Klasa wewnętrzna ***Compar*** implementująca interfejs i udostępniająca metodę compare, która przyjmuje 2 obiekty i zawiera utworzony schemat postępowania w przypadku różnych lub równych obiektów.



Rysunek Implementacja klasy Compar

Klasa ***Point2d*** reprezentująca zbiór punktów w przestrzeni dwuwymiarowej.



Rysunek Implementacja klasy Point2d

# Algorytm Grahama – wyznaczanie otoczki wypukłej

Zapewnia obliczenie otoczki wypukłej zestawu punktów na płaszczyźnie. Działa w czasie O(n log n) i używa O(n) dodatkowej pamięci.

Na początku sortujemy zbiory punktów względem wartości Y następnie po wartości X. Potem bierzemy pierwszą wartość z posortowanego zbioru. Następnie tworzymy listę punktów z wartością kątową miedzy punktem początkowym a każdym kolejnym punktem i sortujemy ją. W kolejnym kroku będziemy rozpatrywali 3 punkty (ozn. A, B, C) zaczynając od tego o najmniejszej wartości współrzędnej x oraz y. Jeśli punkt B leży na zewnątrz prowizorycznego trójkąta zbudowanego na wierzchołku początkowym, oraz wierzchołku A i C to on należy do otoczki wypukłej. Jeśli B leży wewnątrz to B nie należy do otoczki wypukłej - należy usunąć ten punkt ze stosu i cofnąć się o jedną pozycję (o ile jest różna od zera).