Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

ZADANIE 3

Zaawansowane techniki programowania

Prowadzący: dr inż. Jerzy Jaworowski

II stopień - rok 1, semestr 1

wykonanie:

Piotr Adam Tomaszewski

Nr albumu: 104896

Wstęp

Należało zaimplementować program o nazwie $MyClient$, który powinien wykonać udostępnioną przez komponent metodę *register* podając jako parametr właściwy numer zadania oraz numer albumu studenta. Po zakończonej powodzeniem rejestracji program pobiera bliżej nieokreśloną liczbę danych korzystając z metod $hasNext$ i $next$. Następnie z tych danych należało obliczyć moment bezwładności opisanego danymi wejściowymi układu punktów materialnych względem osi obrotu $ax + by + c$.

Organizacja plików

IDataMonitor.java

Interfejs IDataMonitor posiadający w sobie deklarację trzech metod

```
public interface IDataMonitor {
    public boolean register(int hwork, String album);
    public boolean hasNext();
    public double next();
}
```

Rysunek 1 Zawartość interfejsu IDataMonitor

Metoda ***register*** ma za zadanie zarejestrować użytkownika w systemie zwracając $true$ jeżeli proces rejestracji zakończył się poprawnie.

> @param **hwork** - numer zadania
>
> @param **album** – numer albumu studenta
>
> @return - wartość logiczna $true$ albo $false$ w zależności od pomyślności rejestracji

Metoda ***hasNext*** sprawdza, czy istnieją jeszcze dane do pobrania w programie głównym

> @return - wartość logiczna $true$ albo $false$ w zależności od istnienia danych ($false$ jeśli rejestracja nie przebiegła pomyślnie)

Metoda ***next*** pozwalająca na pobranie kolejnych danych w programie głównym

> @return wartość logiczna $true$ albo $false$ w zależności czy dane zostały poprawnie pobrane ($false$ jeśli rejestracja nie przebiegła pomyślnie).

Metoda ***register*** ma za zadanie zarejestrować użytkownika w systemie zwracając *true* jeżeli proces rejestracji zakończył się poprawnie. Jeśli zwróci prawdę to następnie wykonywane są metody ***hasNext*** oraz ***next***, które pobierają ciąg danych wejściowych niezbędnych do obliczeń wykonywanych w głównym programie.

MyClient.java

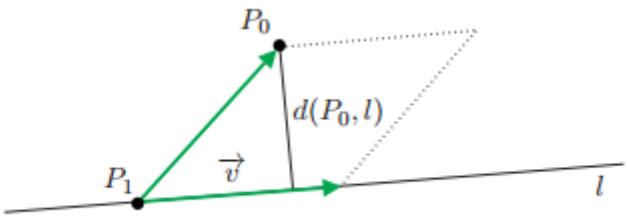
Plik zawierający główną klasę $MyClient$ w której znajduje się główny program. Składa się ona z metody $main$ oraz metod pomocniczych. Dodatkowo zaimplementowano dwie klasy wewnętrzne, z których druga jest rozszerzana przez pierwszą. Klasa zawiera implementacje algorytmu obliczającego moment bezwładności kulek w przestrzeni 3D. Przyjmuje ona parametry a, b, c

będące współczynnikami prostej w przestrzeni, oraz parametry x, y, z będące współrzędnymi punktu wraz z jego $masą$.

```
public class MyClient {
    private double a;
    private double b;
    private double c;
    private final List<Factor> factorList = new ArrayList<Factor>();
```

Rysunek 2 Implementacja klasy MyClient

Pierwszym zadaniem było wyznaczenie odległości punktu P_0 od prostej l. Zauważono że ta odległość może być wysokością równoległoboku o wierzchołku znajdującym się na prostej. Pole równoległoboku utworzonego przez wektory $\vec{v}$ oraz $\overrightarrow{P_1P_0}$ jest równe długości iloczynowi wektorów $\left|\overrightarrow{P_1P_0}\ x\ \vec{v}\right|$. Aby wyznaczyć wysokość należy pole równoległoboku podzielić przez długość wektora $\vec{v}$. Stąd ostateczny wzór na odległość punktu od prostej wyraża się $d(P_0, l) = \frac{\left|\overrightarrow{P_1P_0}\ x\ \vec{v}\right|}{|\vec{v}|}$.



Rysunek 3 Odległość punktu P_0 od prostej l[1]

Klasa ***Point*** reprezentująca zbiór punktów x, y, z w przestrzeni trójwymiarowej.

[1] Źródło rysunku: http://home.agh.edu.pl/~gora/algebra/Wyklad12.pdf

```
private class Point{
    protected double x;
    protected double y;
    protected double z;

    public Point(double x, double y, double z){
        this.x = x;
        this.y = y;
        this.z = z;
    }

    protected double getX() {
        return x;
    }

    protected void setX(double x) {
        this.x = x;
    }

    protected double getY() {
        return y;
    }

    protected void setY(double y) {
        this.y = y;
    }
```

Rysunek 4 Implementacja klasy Point

Klasa Faktor dziedzicząca pola i metody po klasie Point reprezentująca zbiór punktów x, y, z w przestrzeni 3D oraz masę punktu.

```
private class Factor extends Point{
    private double t;

    public Factor (double x, double y, double z, double t){
        super(x,y,z);
        this.t = t;
    }

    private double getT() {
        return t;
    }

    private void setT(double t) {
        this.t = t;
    }
}
```

Rysunek 5 Implementacja klasy Factor

Metoda statyczna ***main***, która wywoływana jest jako pierwsza, gdy uruchamiamy program. Inicjuje połączenie z komponentem $ejb - project$, który posiada dane do zaimportowania przez program główny

@param **args** - tablica elementów typu String

```
public static void main(String[] args)throws Exception{
    MyClient mc = new MyClient();
    String connect = "java:global/ejb-project/DataMonitor!"
                        +"pl.jrj.data.IDataMonitor";

    InitialContext con = new InitialContext();
    pl.jrj.data.IDataMonitor monit = (pl.jrj.data.IDataMonitor)
    con.lookup(connect);

    if(monit.register(3, "104896")){
        mc.addData(monit);
        mc.prepare();
    }
}
```

Rysunek 6 Implementacja metody main

Metoda ***addData*** pozwalająca pobierać dane z komponentu $ejb - project$ i zapisywać dane w parametrach klasy i listy.

@param **monit** - odniesienie do obiektu komponentu

@param **i** - deklaracja zmiennej potrzebnej do iteracji po liście

```
private void addData(IDataMonitor monit, int i){
    List<Double> dataMonit = new ArrayList<Double>();
    while (monit.hasNext()){
       dataMonit.add(monit.next());
    }
    a = dataMonit.get(0);
    b = dataMonit.get(1);
    c = dataMonit.get(2);
    int dataSize = dataMonit.size();
    for (i=3;i<dataSize;i+=4){
        factorList.add(
            new Factor(dataMonit.get(i),dataMonit.get(i+1),
                       dataMonit.get(i+2),dataMonit.get(i+3)));
    }

}
```

Rysunek 7 Implementacja metody addData

Metoda ***prepare*** przygotowująca współrzędne punktu wektora v do obliczenia jego długości. Do obliczenia odległości punktów od prostej wykorzystywana jest metoda równoległoboku, w której szukana odległość to wysokość równoległoboku.

```
private void prepare(){
    Point v0 = new Point(0,0,c);
    Point vk = new Point(1,1,a+b+c);
    Point v = coordinates(v0,vk);
    System.out.println(v.getX()+","+v.getY()+","+v.getZ());
    double suma = sumInertia(0,v0,v,0,0);

    System.out.format(Locale.ENGLISH,"%.5f",suma);
}
```

Rysunek 8 Implementacja metody prepare

Metoda ***coordinates*** zwraca współrzędne wektora znając jego współrzędną początkową i końcową przy wykorzystaniu wzoru matematycznego

@param **v0** - współrzędne punktu początkowego wektora

@param **vk** - współrzędne punktu końcowego wektora

@return - współrzędne wektora

```
private Point coordinates(Point v0, Point vk){
    return (new Point(
            vk.getX()-v0.getX(),
            vk.getY()-v0.getY(),
            vk.getZ()-v0.getZ()));
}
```

Rysunek 9 Implementacja metody coordinates

Metoda ***multiplyVectors*** pozwalająca wyznaczyć iloczyn wektorowy dwóch wektorów, które są dowolnymi wektorami w przestrzeni trójwymiarowej.

@param **p** - współrzędne wektora pierwszego

@param **v** - współrzędne wektora drugiego

@return - iloczyn wektorowy dwóch wektorów

```
private Point multiplyVectors(Point p, Point v) {
   return new Point(
           (p.getY()*v.getZ())-(v.getY()*p.getZ()),
           (v.getX()*p.getZ())-(p.getX()*v.getZ()),
           (p.getX()*v.getY())-(v.getX()*p.getY())
   );
}
```

Rysunek 10 Implementacja metody multiplyVectors

Metoda ***lengthVector*** oblicza długość wektora z jego współrzędnych wzorem matematycznym

@param **mnozenie** - współrzędne wektora w przestrzeni 3D

@return - długość wektora

```
private double lengthVector(Point mnozenie) {
    return Math.sqrt((mnozenie.getX()*mnozenie.getX())
            +(mnozenie.getY()*mnozenie.getY())
            +(mnozenie.getZ()*mnozenie.getZ())
    );
}
```

Rysunek 11 Implementacja metody lengthVector

Metoda ***sumInertia*** liczy dla każdego punktu o masie m odległość tych elementów od prostej będącej osią obrotu. Moment bezwładności jest iloczynem masy punktu i kwadratem jego odległości od osi obrotu. Zadanie polega na zsumowaniu wszystkich momentów bezwładności i wyniku na ekran.

@param **suma** - wartość początkowa zmiennej suma

@param **v0** - współrzędna punktu początkowego wektora V

@param **v** - współrzędna wektora V

@param **licznik** - wartość początkowa zmiennej licznik

@param **mianownik** - wartość początkowa zmiennej mianownik

@return - suma wartości momentów bezwładności punktów

```
private double sumInertia (double suma, Point v0, Point v,
        double licznik, double mianownik){
    Point p;
    for (Factor xyzt : factorList){
      p = coordinates(v0,new Point(xyzt.getX(),xyzt.getY(),xyzt.getZ()));
      Point mnozenie = multiplyVectors(p,v);
      licznik = lengthVector(mnozenie);
      mianownik = lengthVector(v);
      double wynik = licznik/mianownik;
      suma+=(wynik*wynik)*xyzt.getT();
    }
    return suma;
}
```

Rysunek 12 Implementacja metody sumInertia