I. Beadandó

Habzda Fruzsina (XNUHTE)

Tartalomjegyzék

Feladat

Rögzítsen a síkon egy pontot, és töltsön fel egy gyűjteményt különféle szabályos (kör, szabályos háromszög, négyzet, szabályos hatszög) síkidomokkal! Határozza meg a legkisebb téglalapot, amely lefedi az összes síkidomot és oldalai párhuzamosak a tengelyekkel! Minden síkidom reprezentálható a középpontjával és az oldalhosszal, illetve a sugárral, ha feltesszük, hogy a sokszögek esetében az egyik oldal párhuzamos a koordináta rendszer vízszintes tengelyével, és a többi csúcs ezen oldalra fektetett egyenes felett helyezkedik el. A síkidomokat szövegfájlból töltse be! A fájl első sorában szerepeljen a síkidomok száma, majd az egyes síkidomok. Az első jel azonosítja a síkidom fajtáját, amit követnek a középpont koordinátái és a szükséges hosszúság. A feladatokban a beolvasáson kívül a síkidomokat egységesen kezelje, ennek érdekében a síkidomokat leíró osztályokat egy közös ősosztályból származtassa!

A Program bemenete

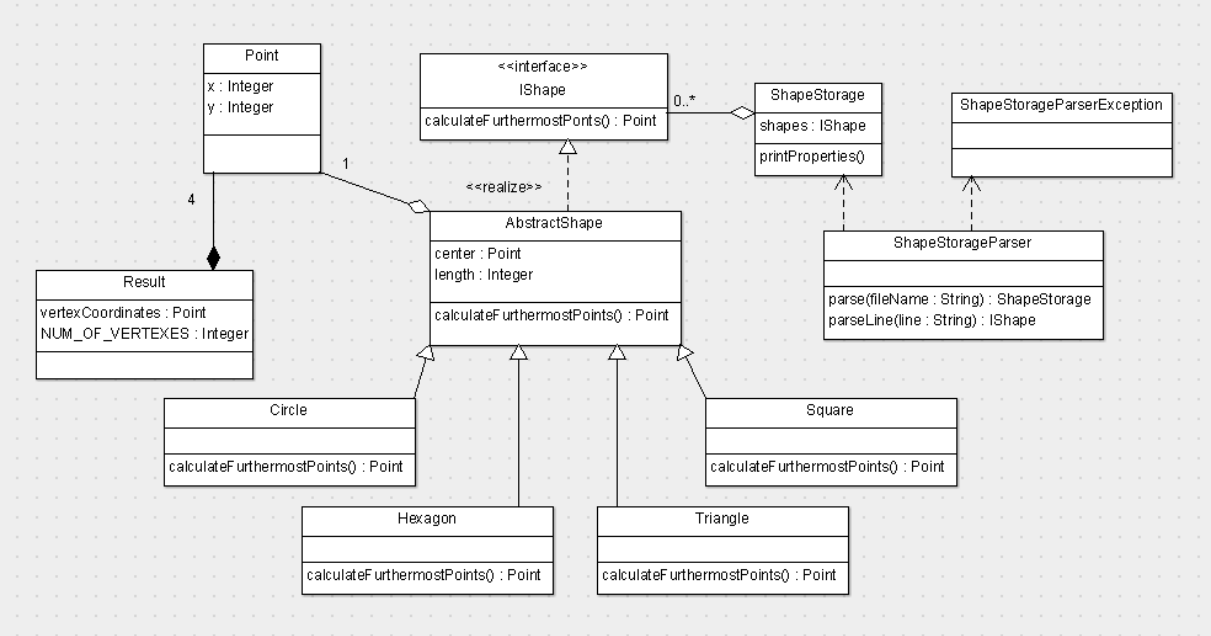
#	Adat	Magyarázat
1.	*Alakzat$_1$ x y hossz*	***Alakzat (szöveg):*** lehet háromszög („tri”), négyzet („sqr”), kör („cir”),
2.	*Alakzat$_2$ x y hossz*	hatszög („hex”) ***x (egész):*** középpont x koordinátája ***y (egész):*** középpont y koordinátája ***hossz (egész):*** oldalhossz, vagy kör esetében sugár hossz
…	…	

Tervezés

Specifikáció

Az alakzatok leírásához bevezettem egy interface-t egy absztrakt osztályt, és négy ezekből származtatott osztályt (*IShape, AbstractShape, Circle, Triangle, Square, Hexagon*). Mivel koordináta rendszerben dolgozunk, és koordinátákat kérünk be, bevezettem a pont (*Point*) osztályt is. A beolvasott alakzatokat egy arra alkalmas osztályban tárolom (*ShapeStorage*), míg a beolvasást is külön osztállyal végzem (*ShapeStorageParser*). A beolvasás folyamán kivételek léphetnek fel, amihez létrehoztam egy az *Exception*-ből származtatott osztályt (*ShapeStorageParserException*). Az eredménynek is csináltam egy külön osztályt, ami csak a csúcsok koordinátáit tartalmazza. (Azért nem az alakzatokból származtattam, mert egyrészt ez nem szabályos alakzat, másrészt pedig semmi egyebet nem szeretnénk vele csinálni, mint hogy létrehozzuk.)

Osztály diagram



Megvalósítás

Főprogram

A főprogramban beolvasom az alakzatokat egy szöveges fájlból a *ShapeStorageParser* segítségével, mely megtölti *shapeStorage*-ot. Majd kiíratom a beolvasott alakzatok adatait, mely a megoldás szempontjából nem szükséges. Ezután a *countResult()* metódus meghívásával kiszámolom az eredményt, azaz az összes alakzatot lefedő téglalap csúcsának koordinátáit, és ki is iratom azt.

```
ShapeStorage shapeStorage;
ShapeStorageParser shapeStorageParser = new ShapeStorageParser();

shapeStorage = shapeStorageParser.parse("tobbdb.txt");

System.out.println("Properties: ");
shapeStorage.printProperties();
System.out.println();

Result result = countResult(shapeStorage);
System.out.println(result.toString());
```

A főprogram tehát 3 metódusból áll:

```
• public static void main(String[] args)
• private static Result countResult(ShapeStorage shapeStorage)
• private static void compareMaxes(ShapeStorage shapeStorage, Float[]
  maxesOfAxes, Integer index)
```

countResult()

paraméter1: ShapeStorage
visszatérési érték: Result
leírás: végigiterál a ShapeStorage-on, és mindegyik elemére meghívja a *compareMaxes()* metódust

compareMaxes()

paraméter1: ShapeStorage
paraméter2: valós számokból álló tömb
paraméter3: egész szám
visszatérési érték: -
leírás: a valós számokból álló tömb elemeit hasonlítja össze, és írja felül a ShapeStorage egész számadik elemén meghívott *calculateFurthermostPoints()* által visszaadott tömb elemeivel

Érdekesebb metódusok

Az alakzatok esetében mindnél más a *calculateFurthermostPoints()*, ami kiszámolja a legszélsőségesebb fekvésű pontjait, 4 irányban.

Circle:

(kiszámolása triviális)

```
@Override
public Float[] calculateFurthermostPoints(){
    Float[] res = new Float[4];
    res[0] = (center.getY() + length); //fel
    res[1] = (center.getY() - length); //le
    res[2] = (center.getX() + length); //jobbra
    res[3] = (center.getX() - length); //balra
    return res;
}
```

Triangle:

(A szabályos háromszög magassága az oldal hosszának √3 / 2 szerese. Tehát a felső pontja a háromszögnek a középpont és a magasság 2/3-ának összege, alsó oldala pedig a középpont és a magasság 1/3-ának különbsége. Az x tengelyen lévő legtávolabbi pontok pedig a középpont és az oldal hossz felének összege, illetve különbsége.)

```
@Override
public Float[] calculateFurthermostPoints(){
    Float[] res = new Float[4];
    res[0] = (center.getY() + ((float)(length)*(float)(Math.sqrt(3)) / 2) /
3 * 2); //fel
    res[1] = (center.getY() - ((float)(length*Math.sqrt(3)) / 2) / 3); //le
    res[2] = (center.getX() + ((float)(length) / (float)(2))); //jobbra
    res[3] = (center.getX() - ((float)(length) / (float)(2))); //balra
    return res;
}
```

Square

(kiszámolása triviális)

```
@Override
public Float[] calculateFurthermostPoints(){
    Float[] res = new Float[4];
    res[0] = (center.getY() + ((float)(length) / (float)(2))); //fel
    res[1] = (center.getY() - ((float)(length) / (float)(2))); //le
    res[2] = (center.getX() + ((float)(length) / (float)(2))); //jobbra
    res[3] = (center.getX() - ((float)(length) / (float)(2))); //balra
```

```
    return res;
}
```

Hexagon

(Mivel a hatszög felbontható egyenlő szárú háromszögekre, melyeknek csúcsai pont a hatszög középpontjánál találkoznak, így a szab. háromszög magasságai (hossz * √3 / 2) határozzák meg a legtávolabbi pontokat.)

```
@Override
public Float[] calculateFurthermostPoints(){
    Float[] res = new Float[4];
    res[0] = center.getY() + ((((float)(length)*(float)(Math.sqrt(3)))/2));
//fel
    res[1] = center.getY() - ((((float)(length)*(float)(Math.sqrt(3)))/2));
//le
    res[2] = center.getX() + ((((float)(length)*(float)(Math.sqrt(3)))/2));
//jobbra
    res[3] = center.getX() - ((((float)(length)*(float)(Math.sqrt(3)))/2));
//balra
    return res;
}
```

Tesztelés

Elvégzett tesztesetek

- *countFurthermostPoints()* metódusok helyességének tesztelése (legtávolabbi pontok kiszámolásának helyessége (minden alakzatnál)
- rossz fájl elérési útvonallal
- üres fájllal (ures.txt)
    - hiba dobás
- olyan alakzatot tartalmazó fájllal, mely nincs a feladatban (hibas.txt)
    - hiba dobás
- negatív hossz megadás (negativhossz.txt)
    - abszolút értékét veszi
- kiterjedés nélküli alakzatok megadása (nullkiterjedes.txt)
    - ez esetben is kezeli őket, mit alakzat
- egy alakzatos fájllal (egydb.txt)
    - beolvasás helyességének megítélése
- több alakzatos fájllal (tobbdb.txt)
    - mindig a legtávolabbi pontokból kerül ki a végeredmény (helyes *countResult()* és *compareMaxes()* metódus)

1. teszteset: Triangle-countFurthermostPoints metódus

Bemenet – (triangle.txt) *középpont, 1 hosszú*
tri 0 0 1
Kimenet
0.57735026

-0.28867513 0.5 -0.5

2. teszteset: Circle-countFurthermostPoints metódus

Bemenet – (circle.txt) *középpont, 1 hosszú*
cir 0 0 1
Kimenet
1.0 -1.0 1.0 -1.0

3. teszteset: Square-countFurthermostPoints metódus

Bemenet – (square.txt) *középpont, 1 hosszú*
sqr 0 0 1
Kimenet
0.5 -0.5 0.5 -0.5

4. teszteset: Hexagon-countFurthermostPoints metódus

Bemenet – (hexagon.txt) *középpont, 1 hosszú*
hex 0 0 1
Kimenet
0.8660254 -0.8660254 0.8660254 -0.8660254

5. teszteset: Hibás elérési útvonal megadás

Bemenet – (pl.: qwerty/asd.txt)
Kimenet
A fajl nem talalhato a qwerty/asd.txt utvonalon Properties: Exception in thread "main" java.lang.NullPointerException at Boot.main(Boot.java:21) Process finished with exit code 1

6. teszteset: Üres fájl megadása

Bemenet – (ures.txt)
Kimenet
Properties: Egyetlen alakzat sincs amit vizsgalni lehetne. Ha ugy ertelmezzuk a feladatot, a megoldásnak az osszes csucsa a (0.0, 0.0) pontban van. Exception in thread "main" java.lang.NullPointerException at Boot.main(Boot.java:25) Process finished with exit code 1

7. teszteset: Olyan fájl megadása, amelyben nem létező alakzat van

Bemenet – (hibas.txt)
tri 2 4 3 hi 5 2 1 cir 6 -1 2
Kimenet
Nem letezo alakzat Properties: Exception in thread "main" java.lang.NullPointerException at Boot.main(Boot.java:21) Process finished with exit code 1

8. teszteset: Az alakzatok konstruktorának tesztelése (negatív hossz esetén)

Bemenet – (negativhossz.txt)
tri -1 -3 -5
Kimenet
Properties: Triangle{center=(-1.0, -3.0), length=5}

9. teszteset: Alakzatok nulla hosszúsága esetén

Bemenet – (nullkiterjedes.txt)
tri 0 0 0 sqr -1 2 0 hex -4 -2 0
Kimenet
Result: vertexCoordinates=[(0.0, 2.0), (-4.0, 2.0), (-4.0, -2.0), (0.0, -2.0)]

10. teszteset: Egy alakzat beolvasása

Bemenet – (egydb.txt)
hex -4 3 6
Kimenet
Hexagon{center=(-4.0, 3.0), length=6}

Result: vertexCoordinates=[(1.1961522, 8.196152), (-9.196152, 8.196152), (-9.196152, -2.1961522), (1.1961522, -2.1961522)]

11. teszteset: Több alakzat beolvasása

Bemenet – (tobbdb.txt)
hex 2 3 2 tri -2 -4 3 sqr 3 1 5 hex 1 0 4 cir 7 3 3 cir -5 -2 4 tri -3 5 3 sqr 6 -2 3 hex 1 0 4 cir 7 3 3 hex 2 2 2 tri -1 -4 3 tri -1 -7 2
Kimenet
Result: vertexCoordinates=[(10.0, 6.732051), (-9.0, 6.732051), (-9.0, -7.57735), (10.0, -7.57735)]

További tesztelési terv

Tesztesetek a kód alapján (fehér doboz tesztelés)

1) Extrém mennyiségű, méretű alakzat beolvasása, kezelése.

Habzda Fruzsina
XNUHTE
1. beadandó