Programozási technológia 1.

Első beadandó – Dokumentáció

Kósa Réka

SOEJEO

Eötvös Loránd Tudományegyetem

Informatikai Kar

2016/17/2

Nagy Sára

A feladat leírása

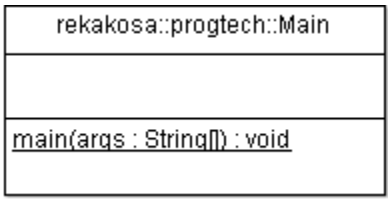
> *Rögzítsen a síkon egy pontot, és töltsön fel egy gyűjteményt különféle szabályos (kör, szabályos háromszög, négyzet, szabályos hatszög) síkidomokkal! Keresse meg, melyik síkidom van legközelebb a ponthoz! Kör esetén a közelséget a körvonaltól vett távolság adja meg, ha a pont a körön kívül van. Ha a pont a körön belül helyezkedik el, akkor a távolságukat nullának tekintjük. 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!*

Megoldási terv

A kétfajta alakzatot (`Circle`, `Polygon`) egy közös `Shape` ősosztályból származtatjuk (ezek mind a `Point2D`, pontokat reprezentáló osztályt veszik alapul), ám a ponttól vett távolságukat eltérően határozzuk meg. A program futása a Main osztállyal indul, ami a `Parser` segítségével beolvassa és megfelelő formátumúvá alakítja az adatokat. Ezt követően fajtától függően egyesével kiszámoljuk, mekkora távolságra van egy adott alakzat a referenciaponttól, mindig megjegyezve az addigiak közül a minimálist. Ha az aktuális elem egy kör, akkor a távolság-meghatározás egyszerűen történik: a körvonaltól vett távolságot nézzük, sokszög esetén azonban nem hagyatkozhatunk szimplán a köré írt kör sugarára. Előfordulhat ugyanis, hogy veszünk egy szabályos sokszöget a köré írt körével, egy kört, valamint a pontot e két körtől ugyanolyan távol helyezzük el. Ha az előbbi módszert alkalmaznánk, akkor egyenlő távolságúnak ítélnénk meg a köröket, viszont lehet, hogy a pont a sokszögnek épp egyik oldala felett található, épp ezért a távolságuk is nagyobb kell, hogy legyen. Így a következőhöz folyamodunk: ha sokszöggel van dolgunk, első körben megvizsgáljuk, hogy a pont beleesik-e valamely, a sokszög egyik oldala és annak a végpontjain (a csúcsok) átmenő merőlegesek által meghatározott, nyílt tartományba. Amennyiben igen, akkor a távolság a megfelelő oldal és a pont távolsága lesz. Ha nem, akkor ellenőrizzük, hogy a pontunk és az alakzat középpontjának a távolsága nagyobb-e, mint a sugár. Ha nem, az azt jelenti, hogy a pont a sokszöglapon található, hiszen a tartományozással kizártunk már más eseteket; ha nem, akkor a pont két szomszédos oldal közös csúcson átmenő merőlegesei által kijelölt holttérben kell, hogy legyen, tehát ekkor a távolság a csúcstól mért távolsággal lesz egyenlő. Ha végigértünk az összes objektumon, kiíratjuk a legközelebbi síkidom adatait.

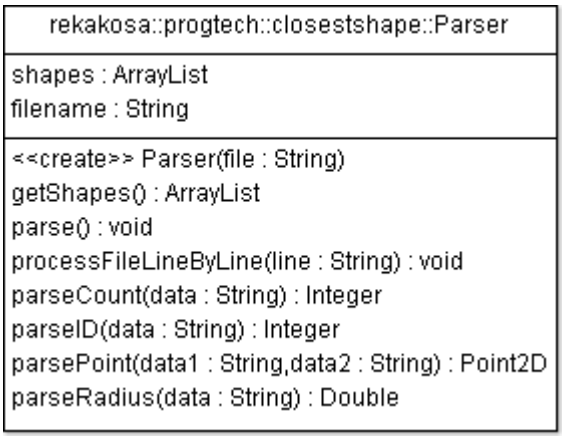
Osztályok

Main



A main függvény bekér a felhasználótól egy fájlnevet, illetve két koordinátát, ezek lesznek a referenciapont koordinátái. Ezeket átadja a `Parser` osztálynak, ami elkészíti a megfelelő metódusával a pontot, a `Circle` vagy `Polygon` objektumokhoz szükséges paramétereket, majd létre is hozza azokat. Miután ez megvan, egy ciklus végigiterál a `Shape`-ekből álló gyűjteményen, megméri minden egyes alakzat távolságát a ponttól, kiválasztja ezek közül a legkisebbet, és kiírja az ehhez tartozó adatokat.

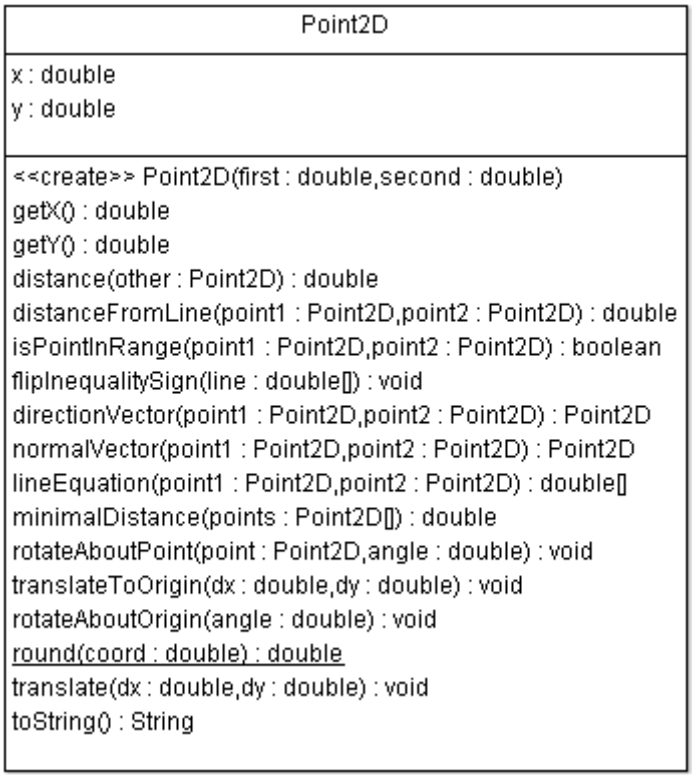
Parser



A `Parser` osztálynak 3 adattagja van, amelyekben a visszatérési értékeket és megadott fájlnevet tároljuk. A `getShapes()` visszaadja a feldolgozott listát, a `getPoint()` az ellenőrzött pontot. A `parse()` metódus végigmegy a fájlon, és az egyes blokkokat külön függvények segítségével olvassa be soronként, egy `processFileLineByLine()` nevű függvénnyel, ezek rendre: `parseCount()`, `parseID()`, `parseRadius()` és `parsePoint()`.

Point2D

A `Point2D` osztály az építőköve a síkidomoknak. Két adattagja van, ezekben rögzítjük az x és az y koordinátát, ezeket a megfelelő getterek adják vissza, illetve rendelkezik egy konstruktorral is, ami beállítja őket. A `distance()` metódus két pont távolságát adja vissza, a `distanceFromLine()` pedig egy pontét és két pontjával megadott egyenesét. A `minimalDistance()` egy pontokból álló tömböt vár, és kiválasztja azt, amelyik a legközelebb van a pontunkhoz, erre később, a `Shape` távolságmetódusánál lesz szükség. A `rotateAboutPoint()` elforgatja a pontot a paraméterül adott pont körül, mindezt úgy, hogy eltolja a két pontot az origóba, így már csak egy origó körüli forgatást kell végeznünk, majd visszatolnunk az eredeti helyére: ezt a `translateToOrigin()`,

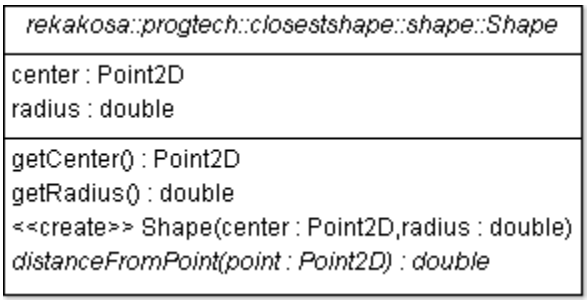


`rotateAboutOrigin()` és `translate()` metódusokkal hajtjuk végre.

Az `isPointInRange()` függvény felelős azért, hogy eldöntse, hogy a paraméterül kapott két pont közül a rajtuk átmenő egyenes és a pontokon keresztül húzott merőlegesek közötti tartományba esik-e a pont. Ez egy nyílt intervallum, vagyis maguk az egyenesek már nem számítanak bele. Ennek érdekében a metódus felír három egyenletet, és a pontok x és y koordinátáinak egymáshoz való viszonya alapján eldönti, hogy melyik egyenes milyen állású, tehát hogy milyen feltételek alapján kerülhet valóban ebbe a tartományba. Ezek egyenlőtlenségeket határoznak meg, ha mindhárom teljesül egyszerre, akkor igazat ad, a pont az egyenesek között lesz. Munkájához segítségül hívja a `normalVector()`, `directionVector()` függvényeket, ezekkel előállítja az egyenesek egyenletét a `lineEquation()` függvényben. A `flipInEqualitySign()` az egyenlőtlenség irányát fordítja meg.

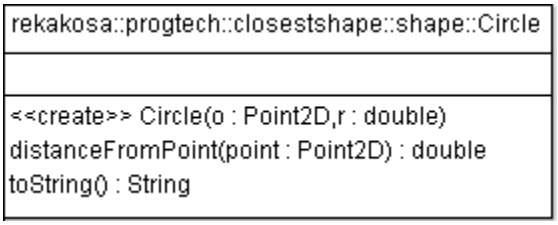
Az osztály tartalmaz még két segédfüggvényt is, az egyik a `toString()`, amely egy pontot szövegesen megjelenítve ad vissza, a másik pedig a `round()`, ami egy megadott valós számot, jelen esetben koordinátát 4 tizedes pontosságúra kerekít.

Shape



A `Shape` absztrakt osztály tárolja azokat a jellemzőket, amelyek bármelyik alakzatra igazak: a középpontot és a sugarat. Van egy konstruktora, az adattagok getter metódusai, és egy absztrakt `distanceFromPoint()` metódus, aminek a paramétere egy pont, és definíciója más és más a `Shape`-ből származtatott osztályokban.

Circle

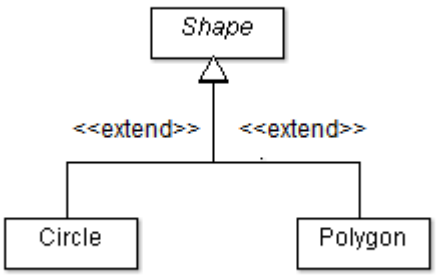


A `Circle` osztály a `Shape`-ből származik, egy kört reprezentál. Adattagjait a szülő osztálytól örökli. A `distanceFromPoint()` metódust felüldefiniálja oly módon, hogy ha a középpont és a megadott pont távolsága nagyobb, mint a sugár, akkor visszaadja ezek különbségét ebben a sorrendben, különben pedig 0-t. Szerepel még egy `toString()` metódus, amely egy kör és az adatai szöveges megjelenítésére hivatott.

Polygon

rekakosa::progtech::closestshape::shape::Polygon
numberOfAngles : int vertices : Point2D[]
getNumberOfAngles() : int <<create>> Polygon(center : Point2D,radius : double,angles : int) constructPolygon(center : Point2D,radius : double,numberOfAngles : int) : void distanceFromPoint(point : Point2D) : double toString() : String verticesToString() : String

A `Polygon` szintén a `Shape` gyereke, ez az osztály reprezentálja a szabályos sokszöget. Négy adattagja van, melyek közül kettő, a sugár és a középpont a `Shape`-ből származik, a maradék kettő pedig a szögek száma és a csúcsok koordinátái egy tömbben, ez utóbbihoz tartozik egy getter metódus, ehhez pedig kapcsolódik a `verticesToString()` metódus. Ezekre a követhetőség érdekében van szükség, a `Main`ben hívódnak meg, és a metódus kiírja a csúcsok koordinátáit. Mivel a feladat specifikációja szerint egyik oldal párhuzamos az x tengellyel, és az összes többi csúcs felette helyezkedik el, a sokszög egyértelműen meghatározott. A konstruktorban elmentjük a sugarat, középpontot és szögek számát is, ez után feltöltjük a csúcsok tömbjét a koordinátáikkal. Az imént hivatkozott jóldefiniáltság miatt a legelső csúcs koordinátáit szögfüggvényekkel, az összes többit pedig az óramutató járásával ellentétes, tehát a szokásos körüljárási irányban, forgatásokkal kapjuk. A sokszög szabályossága miatt mindig ugyanakkora szöggel kell forgatnunk, ennek a nagyságát könnyen ki tudjuk számolni, ha a sokszög belső szögösszegét elosztjuk a szögek számával. A `distanceFromPoint()` metódus működése a következő: először is megbizonyosodik arról, hogy beleesik-e a pont valamelyik oldal és a kapcsolódó csúcspontok merőlegesei közé, ehhez pontpárokat vizsgál – ha ez a feltétel teljesül, akkor visszatér az oldal és a pont távolságával. Ha nem, megnézi, hogy a középpont és a referenciapont távolsága kisebb vagy egyenlő-e, mint a sugár – ha ez áll fenn, akkor a sokszögön van a pont, ezért visszatér 0-val. Ha ez sem igaz, akkor az idáig még nem vizsgált részek egyikében, a csúcspontokon áthaladó merőlegesek által közrefogott valamelyik tartományban kell lennie. A vizsgálat során nem tartottuk nyilván, hogy melyik tartományhoz van *közel* a pont, ezért vesszük az összes csúcstól a távolságát, és ezek minimuma lesz a keresett érték. A `Polygon` `toString()` metódusa szögek száma alapján konvertálja szöveggé a sokszöget és adatait.



Tesztelés

A projektben elérhető néhány előre elkészített teszteset. A főprogram először a beolvasandó fájl nevét kéri, erre 10 példát is láthatunk a becsomagolt mappaszerkezeten belül. A `Parser` vesszővel elválasztott adatokat vár, ha nem ilyen fájl érkezik, akkor azt nem tudja kezelni. A fájl első sora tartalmazza a síkidomok számát, utána minden egyes sor a síkidom adatait. A legelső adat a fajtáját azonosítja be: esetünkben 0 jelzi a körre, míg a 3, 4 vagy 6 a három-, négy- vagy hatszögre utalnak. A következő két adat adja a középpont koordinátáit, és az utolsó, negyedik lesz a sugár.

input.txt + beolvasott pont: (-2, 1)

- *több alakzat van, többfélék, mindegyiken kívül helyezkedik el a pont:* a négyzet a legközelebbi

input.txt + beolvasott pont: (0, 0) és input.txt + beolvasott pont: (4, -3)

- *egyik alakzaton belül helyezkedik el a pont (első eset: négyzet, második eset: egyik kör):* 0 a távolság az elvártnak megfelelően

input2.txt + beolvasott pont: (3.5, 2)

- *több alakzat, kizárólag különböző fajtájú sokszögek, minden kívül:* legközelebbi alakzat a háromszög

input3.txt + beolvasott pont: (1, 1)

- *egy alakzat van, a legelső a minimum:* helyes működés

input.txt + beolvasott pont: (4, -2)

- *több alakzat, az utolsó a minimum (az egyetlen):* helyes működés

input4.txt + külső beolvasott pont: (8, 10)

- *több alakzat van egymáson, a sugaraik és a középpontjaik mind megegyeznek egymásával:* a kör lesz a minimum, helyes működés

input5.txt + beolvasott pont: (1, 1)

- *nem a megadott számú alakzat van:* mivel az alakzatok egy `ArrayList`ben vannak tárolva, mindenféleképp lesz elég hely az összes számára

input6.txt + beolvasott pont: (0, 0)

- *több objektum van ugyanakkora távolságra a ponttól:* a legelsőt választja ki

input77.txt

- *nem létezik a megadott file:* hibajelzés, a program működése befejeződik

input7.txt + (0,0)

- *üres a megadott file:* hibajelzés, a program működése befejeződik

input.txt + (123, -4a)

- *a referenciapont nem megfelelő:* hibajelzés, a program működése befejeződik

input8.txt

- *hibás a megadott azonosító:* hibajelzés, a hibás adat sora nem kerül mentésre

input9.txt

- *hibás a középpont:* hibajelzés, a hibás adat sora nem kerül mentésre

input10.txt

- *hibás a megadott sugár:* hibajelzés, a hibás adat sora nem kerül mentésre

Fejlesztési lehetőségek

Későbbi funkciókként beiktatható lenne, hogy bármilyen karakterrel elválasztott adatokkal tudjon dolgozni a program. Több minimum esetén mindet eltárolhatná, hiszen ekkor ugyanolyan közel vannak az objektumok a ponthoz. Hatalmas adatmennyiség esetén végezhetne kizárással járó felméréseket minden egyes távolságmérés után, amelyek biztosan nincsenek közelebb, mint az eddig lemért objektumok, hogy ne kelljen esetleg mindent megmérni.