1. Beadandó

Programozási technológia 1.

Keszei Ábel
ELTE-IK

Feladat leírása

Töltsön fel egy gyűjteményt különféle szabályos (kör, szabályos háromszög, négyzet, szabályos hatszög) síkidomokkal! ***Adja meg azt a síkidomot, amelynek a területe és a kerülete a legkisebb mértékben tér el egymástól!*** 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

Az összes síkidomot egy közös ősosztályból származtatjuk, melynek neve legyen *`SzabályosSíkidom`*. Ez egy absztrakt osztály, hiszen szükségünk van absztrakt metódusokra benne (`terület()` és `kerület()`). Ennek az az oka, hogy az említett metódusok visszatérési értékének kiszámítási módja függ az adott síkidom típusától. Illetve, mivel szabályos sokszögekről beszélünk – eltekintve a körtől – létezik általános képlet mindkét érték (terület és kerület) kiszámítására. Ennek megfelelően az ősosztályból (a *`SzabályosSíkidom`*-ból) származni fog egy *`SzabályosSokszög`* nevű, szintén absztrakt osztály. Ő lesz a szülője a körön kívüli összes szabályos sokszögnek. (Vehetnénk persze a kört is szabályos sokszögnek – hiszen a definíció tulajdonképpen ráillik – viszont így egy végtelen sok csúccsal rendelkező, nulla oldalhosszú szabályos sokszögről beszélnénk, amit kódban nem tudunk kezelni.) A *`SzabályosSokszög`* osztályunk már képes lesz a terület és kerület számítására, csak az oldalak számára lesz szüksége, amit pedig az osztály gyermekei mind beállíthatnak maguknak a nekik megfelelő értékre a saját konstruktorukban.

Terület- és kerületszámítás a *`SzabályosSokszög`* absztrakt osztályban:

Létezik a szabályos sokszögekre használható általános területszámító képlet:

$$T = \frac{1}{2} \cdot A \cdot K$$

$$A = r \cdot \cos\left(\frac{180°}{\mathrm{n}}\right)$$

ahol ***A*** az apotémát (szabályos sokszög beírható körének sugarát), ***K*** és ***T*** a síkidom kerületét és területét, ***r*** a köréírt kör sugarát, ***n*** pedig az oldalak számát jelöli.

Itt láthatjuk, hogy csak az ***r*** és a ***K*** nem rögzített érték (az ***n*** nyilván rögzített, ha adott oldalszámú sokszögről beszélünk). Mindjárt megmutatjuk azt is, hogy a ***K*** is csak az ***r***-től függ.

A kerület kiszámítható, ha vesszük az oldalhossz oldalszámszorosát. Az oldalhossz kiszámítása szabályos sokszögben lehetséges például koszinusztétel használatával:

$$c^2 = a^2 + b^2 - 2ab \cdot cos\gamma$$

vagyis

$$l^2 = 2 \cdot r^2 - 2 \cdot r^2 \cdot \cos\left(\frac{360°}{\mathrm{n}}\right)$$

$$l = \sqrt{2 \cdot r^2 - 2 \cdot r^2 \cdot \cos\left(\frac{360°}{\mathrm{n}}\right)}$$

ahol ***l*** az oldalhosszt, ***r*** a köréírt kör sugarát, ***n*** pedig az oldalak számát jelöli. Láthatjuk, hogy itt is csak az ***r*** az, ami nem rögzített.

Terület- és kerületszámítás a `Kör` osztályban:

Kör esetében a probléma jóval egyszerűbb. Itt ugyanis a terület és kerület kiszámítása a következőképp történik:

$$T = r^2 \cdot \pi$$

$$K = 2 \cdot r \cdot \pi$$

Triviális, hogy csak a sugárra van szükség, hogy kiszámítsuk a két értéket.

Következtetés:

Elegendő minden síkidomhoz négy darab, rendre ugyanolyan típusú adatot bekérnünk: a típus azonosítóját, a középpont x és y koordinátáját, és a síkidom sugarát (melyet a sokszögek esetében majd a köréírt kör sugaraként értelmezünk).

Pontatlansági probléma:

Észrevehető, hogy ez a megoldás nem ad feltétlenül pontos eredményt sok esetben. Felmerül a kérdés, hogy miért alkalmazunk ilyen módszert, ha pontatlan értékek jönnek ki. Ennek két egyszerű oka van:

- Egyrészt szükségtelen a pontosság. Lentebb olvasható a feladat, amely a főprogramban oldandó meg a definiált osztályok segítségével. A feladat szempontjából pedig elhanyagolható a különbség, ami a pontatlanságból adódik, azaz nem változtat magán az eredményen.
- A beadandó célja az általánosítás/specializáció bemutatása osztályokkal, így pedig logikus minél általánosabb rendszerben gondolkodni. (Pontosabb értékekkel dolgozó rendszer kevésbé lenne általános.)

A feladat megoldása:

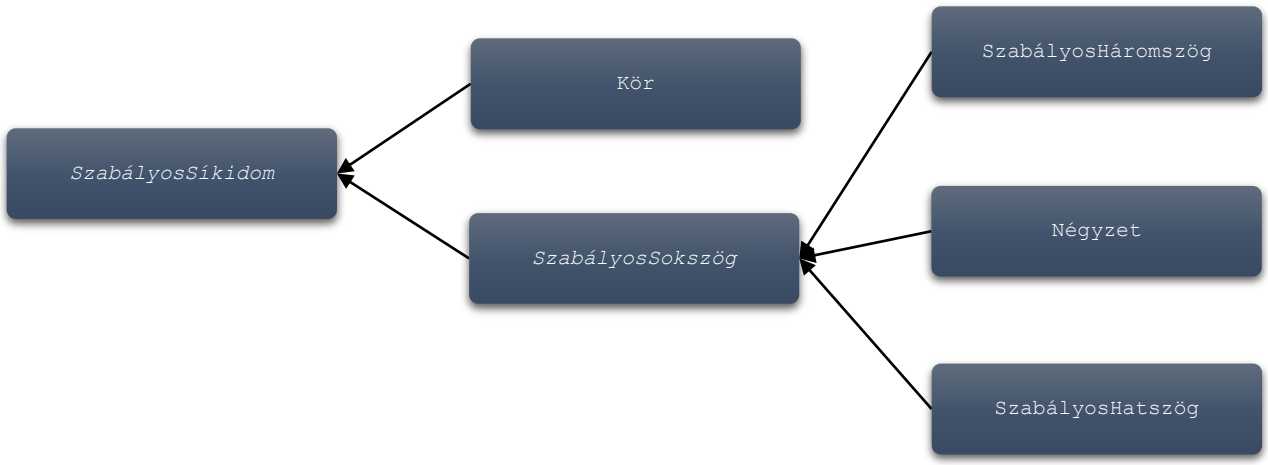
Az eddigiekben az adatok reprezentációjáról és a szükséges értékek kiszámításáról volt szó, de a feladat egy ezekre épülő probléma megoldása. Meg kell mondanunk a beolvasott szabályos síkidomok közül azt, amelyiknek a legkisebb az eltérése a kerülete és a területe között. Szerencsére a Java biztosít eszközt rá, hogy könnyedén kereshessünk minimális elemet egy sorozatban, akárhogyan definiáljuk azon a rendezést. A `Collections` osztály `min()` metódusa visszaadja egy gyűjtemény legkisebb elemét, viszont neki is tudnia kell, hogy ezek

az elemek hogyan viszonyulnak egymáshoz. A rendezés definiálható oly módon egy gyűjteményben, hogy a generikus paraméterében adott osztályban megvalósítjuk a `Comparable` interfészt. Ez megköveteli, hogy definiáljunk egy `compareTo()` metódust, amivel már tud a `Collections.min()` hasonlítást végezni a gyűjtemény elemei között, így képes megtalálni annak minimumát.

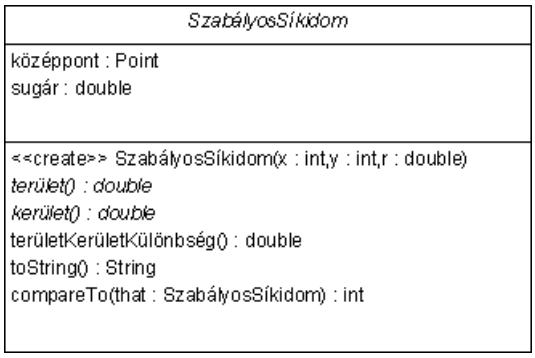
Tehát nincs más dolgunk, mint megadni a `compareTo()` metódus megvalósításában, hogy az az elem nagyobb, amelyiknek nagyobb a kerülete és területe közötti különbség abszolút értéke. Így hívva a `Collections.min()`-t a gyűjteményre, visszakapjuk azt a szabályos síkidomot, amit a feladatleírás alapján kerestünk.

Osztályok és kapcsolataik

Kapcsolatok:

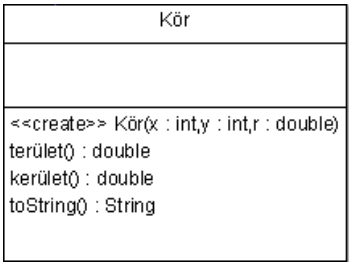


SzabályosSíkidom osztály:



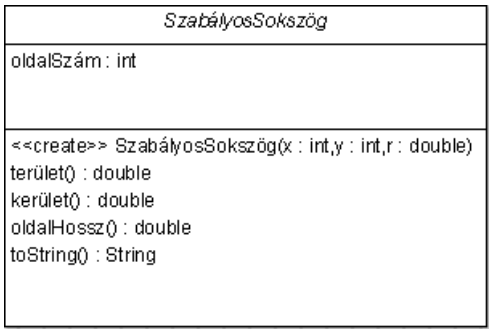
A szabályos síkidomokat reprezentáló absztrakt osztály. Az összes szabályos síkidom rendelkezik egy `középponttal`, illetve azzal a `sugárral`, ami alapján meghatározható egyértelműen a kerülete és a területe (fentebb olvasható, hogy ez mind a körök, mind a szabályos sokszögek esetében miért van így). A körök és a szabályos sokszögek terület és kerület számítása eltér egymástól, így a `terület()` és `kerület()` metódusok absztraktak ebben az osztályban. Viszont a különbségüket számító `területKerületKülönbség()` metódus értelemszerűen nem az. Szükségünk van továbbá egy `toString()` metódusra, hogy az outputra szövegesen kiírhassuk egy-egy síkidom adatait, illetve egy `compareTo()` metódus is kell – mivel az osztály implementálja a `Comparable` interfészt – és persze ne feledkezzünk meg a konstruktorról sem.

`Kör` osztály:



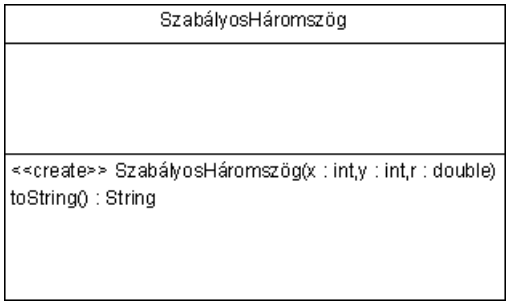
A köröket reprezentáló osztály, amely a *`SzabályosSíkidom`* osztály egy specializációja. Megvalósítja a `terület()` és `kerület()` metódusokat, hogy azok a körökre vonatkoztatott képleteket alkalmazzák az adattagokon. Továbbá felüldefiniálja az általánosabb osztály `toString()` metódusát is, hogy az outputon látszódjon, hogy a kiírt objektum ezen specializáció példánya.

`SzabályosSokszög` osztály:

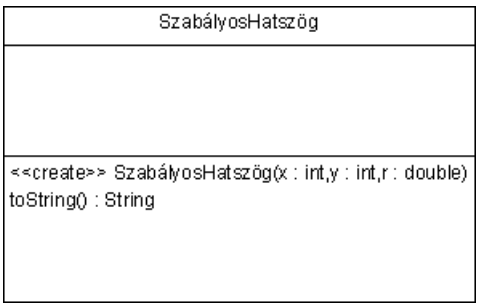


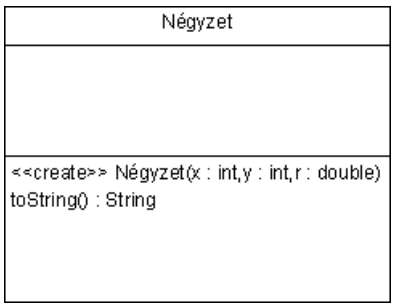
A szabályos sokszögeket reprezentáló absztrakt osztály, amely a szabályos háromszögek és hatszögek, illetve a négyzetek általánosítása. Az általánosabb *`SzabályosSíkidom`* osztályhoz képest bevezet egy új, `oldalSzám` nevű adattagot, hiszen itt már beszélünk ilyen adatról. A `Kör`höz hasonlóan itt is megvalósítjuk a `terület()` és `kerület()` metódusokat, és ahogy az a megoldási tervben szerepelt, ehhez a szabályos sokszögekre általánosan használható képleteket alkalmazzuk az adattagokon. Bevezetünk továbbá egy privát `oldalHossz()` metódust, amely az ismert adatokból megadja az oldalak hosszát, így tudja azt használni a felüldefiniált `toString()`, illetve a kerületszámító metódus is.

`SzabályosHáromszög`, `Négyzet` és `SzabályosHatszög` osztályok:



A *`SzabályosSokszög`* osztály három specializációja, amelyeket a szabályos háromszögek, négyzetek, illetve a szabályos hatszögek reprezentálására használunk. Itt mindössze egy konstruktorbeli értékadás történik, az `oldalSzám` adattag értéke *három*, *négy*, illetve *hat* lesz. Továbbá szintén felüldefiniáljuk a `toString()` metódust, hogy a kiírás tükrözze, melyik specializációról van szó.





Tesztesetek

Az összes teszteset .txt formátumban megtalálható a projekt könyvtár **`teszt`** könyvtárában. NetBeans környezetből futtatva a programot az alapértelmezett munkakönyvtár a projekt könyvtára lesz, így a program listázni fogja az itt is felsorolt teszteseteket.

Érvényes tesztesetek

- **`teszt01.txt`:** tartalmazza az összes típusú síkidomot, szerepel negatív koordinátájú és origóban elhelyezkedő középpontú síkidom is közöttük, van egész és van tizedestört is megadva sugárként. Nem ad a program hibát, hiszen minden adat érvényes.
- **`teszt02.txt`:** a fájlban több síkidom van felsorolva, mint azt az első sor jelzi. Ez a tévedés nem teszi végrehajthatatlanná a feladatot. Ha viszont megvizsgálnánk, hogy valóban annyi síkidom van-e a fájlban, mint az első sorban lévő szám, azt már felesleges lenne odaírni. Így a program a megadott mennyiségű sorig olvas.
- **`teszt03.txt`:** a fájlban nulla darab síkidom van felsorolva (ezt az első sor jelzi, tehát helyes a fájl formátuma). A listázó metódus jelzi, hogy nincsenek síkidomok, a minimumkeresés pedig nem talál minimumot.
- **`teszt04.txt`:** a fájlban található több olyan elem is, amelynek minimális az eltérés a kerülete és a területe között. A program a minimum első előfordulásával tér vissza.

Érvénytelen tesztesetek

- **`teszt05.txt`:** üres tesztfájl. Formátumhibával leáll a feldolgozás.
- **`teszt06.txt`:** a fájlban kevesebb síkidom van felsorolva, mint azt az első sor jelzi. Formátumhibával leáll a feldolgozás.
- **`teszt07.txt`:** a fájl első sorában számként nem értelmezhető adat van. Formátumhibával leáll a feldolgozás.
- **`teszt08.txt`:** a fájlban található egy síkidom, amelynek típusának megadásánál számként nem értelmezhető adat van. Formátumhibával leáll a feldolgozás.
- **`teszt09.txt`:** a fájlban található egy síkidom, amely középpontjának x koordinátájának megadásánál számként nem értelmezhető adat van. Formátumhibával leáll a feldolgozás.
- **`teszt10.txt`:** a fájlban található egy síkidom, amely középpontjának y koordinátájának megadásánál számként nem értelmezhető adat van. Formátumhibával leáll a feldolgozás.
- **`teszt11.txt`:** a fájlban található egy síkidom, amely sugarának megadásánál számként nem értelmezhető adat van. Formátumhibával leáll a feldolgozás.
- **`teszt12.txt`:** a fájl egyik sora kevesebb adatot tartalmaz, mint a szükséges 4, az utána következő pedig többet. A program viszont nem veszi a következő sor első elemét úgy, mintha az a sor utolsó eleme volna, helyette formátumhibával leáll a feldolgozás.
- **`teszt13.txt`:** a fájl egyik sora ismeretlen azonosítójú síkidomot tartalmaz. A program figyelmeztet a helytelen síkidom azonosítóról, majd leáll a feldolgozás.
- **`teszt14.txt`:** a fájl egyik sorában negatív sugarú síkidom szerepel. Formátumhibával leáll a feldolgozás.

- **`teszt15.txt:`** a fájl egyik sorában nulla sugarú síkidom szerepel. Formátumhibával leáll a feldolgozás.

Mellékelt lényeges fájlok

- A **`dist/javadoc`** könyvtárban megtalálható a NetBeans által generált HTML formátumú JavaDoc.
- Az **`src`** könyvtár tartalmazza a program forráskódját.
- A **`teszt`** könyvtár tartalmazza a tesztesetekhez tartozó tesztfájlokat.
- A projektmappa gyökerében található ez a dokumentáció több formátumban, illetve az ArgoUML-ben elkészített, osztálydiagramokat és azok kapcsolatait tartalmazó fájl.