Java heterogén kollekció

Készítette: Goldschmidt Balázs, BME IIT, 2019.

I. Kaszinó

*Egy számítógéppel felügyelt kaszinó működését szeretnénk modellezni. A kaszinóban **játékos**ok ülnek **asztalok**nál. Az **asztalok** (Asztal) nyilvántartják a pillanatnyi tétet, és az asztalnál ülő **játékos**okat, valamint azt, hogy az adott játék hányadik körében járunk. Egy asztalnál maximum 10 játékos ülhet, de lehet üres asztal is. Bármelyik játékos bármelyik asztalhoz leülhet és ott a leülés sorrendjében játszhat. A játék során az asztalba épített számítógép **kör**önként, sorban felszólítja a játékosokat, hogy **lép**jenek. Egy játékos a játék fajtájától függően többféleképpen léphet: passzolhat, tétet emelhet, stb. Az asztal a játék elején egy véletlen célértéket tárol (**goal**). Ha a tét túllépi ezt az értéket, a játéknak vége. Amelyik játékosnál ez bekövetkezik, nyer, ha a tét kevesebb, mint 10%-kal lépi túl goal-t, egyébként veszít. A többi játékos mind veszít.*

***Többféle játékos lehet**, amelyek eltérő stratégiát alkalmaznak: például kezdő (Kezdo) játékos, aki minden páratlan körben passzol, a többiben eggyel emeli a tétet, vagy egy minden körben passzoló robot (Robot) játékos. Minden játékos nyilvántartja, hogy éppen melyik asztalnál ül. A rendszerben az alábbi műveleteket kell megvalósítani:*

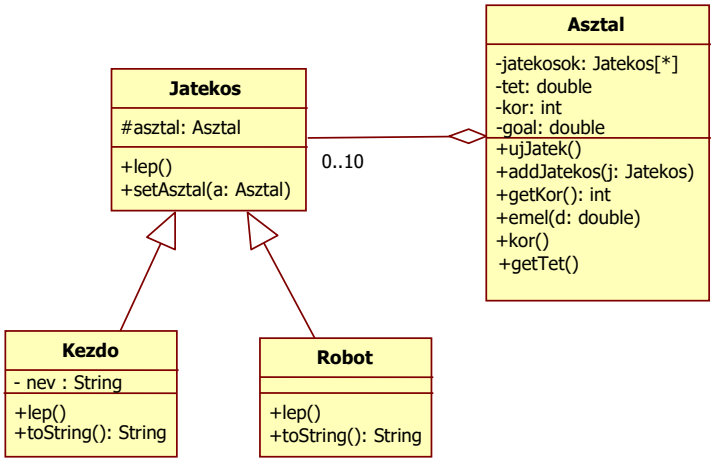
Asztal:

1. *Új játékos csatlakozása az asztalhoz (**addJatekos**). Ilyenkor a játékosban beállítja, hogy melyik asztalnál ül. Jelezze, ha az asztal megtelt!*
2. *Játék kezdése az asztalnál (**ujJatek**). Ennek hatására a tét és a körszámláló nullázódik, a célérték egy véletlen értékre áll (0-100 között).*
3. *Kör végrehajtása (**kor**), melyben minden asztalnál ülő játékos egyet lép. A kör végén az asztal kiírja a tét aktuális értékét. Amikor vége a játéknak, kiírja, hogy hányas sorszámú játékos nyert. Újabb hívásokra, amíg új játék nem kezdődik, azt írja ki, hogy „Vége a játéknak.”*
4. *Kör sorszámának lekérdezése (**getKor**).*
5. *Tét emelése (**emel**) és lekérdezése (**getTet**)*

Játékos:

6. *Lép (**lep**). A játékos ilyenkor megkérdezheti az asztalt, hogy hányadik körnél tartanak, mennyi az aktuális tét, vagy éppen szólhat az asztalnak, hogy emelje a tétet.*

A feladathoz az alábbi osztálydiagram készült (láthatóság: - private, # protected, + public).



1 Aggregáció

a) Implementáljuk az **Asztal** és a **Jatekos** osztályt!
Érdemes először két üres osztályt létrehozni a fenti nevekkel, és utána elkezdeni felvenni az attribútumokat és a metódusokat.
   - A *Jatekos* lep metódusa kiírja, hogy hanyadik körnél járunk, és mekkora a tét.
   - Az *Asztal* a *goal*-t a *Random* osztály segítségével (*nextDouble()* metódus) állítsa be.

b) Készítsünk az osztályok mellé egy **Main** nevű osztályt is, amelyik a **main()** metódust tartalmazza. Ez a metódus hozzon létre egy asztalt, 3 **Jatekos**t, és ültesse a játékosokat az asztalhoz.

c) Játszassunk az asztalnál 3 kört!

2 Heterogén kollekció

a) Implementáljuk a **Kezdo** osztályt, amelyik a **Jatekos** leszármazottja! A nevét konstruktorban lehesen állítani, a **toString()**-gel lekérdezni. A lépésekkor írja ki a nevét, hogy hányadik körnél jár, valamint minden páratlan körben passzoljon, a többiben eggyel emelje a tétet! A név kiíratásakor használjuk a felüldefiniált **toString()** metódust is!

b) Implementáljuk a **Robot** osztályt, amelyik a **Jatekos** leszármazottja! A lépésekkor írja ki, hogy „Robot”, és hogy hányadik körnél jár! A „Robot” szót a felüldefiniál **toString()** metódusból kapjuk meg!

c) Oldjuk meg, hogy a **Jatekos** osztályból ne lehessen példányt létrehozni!

d) A **main()** metódusban **Jatekos**ok helyett két kezdő és egy robot játékost hozzunk létre, őket ültessük az asztalhoz, és így játszassunk 3 kört!

3 Statikus tagok

a) Módosítsuk úgy a **Robot** osztályt, hogy minden robotnak legyen egy saját azonosítója!

b) Az azonosítót úgy inicializáljuk a konstruktorban, hogy a robotok azonosítói növekvő számsort alkossanak!

c) A robotok **toString()** metódusa a „Robot” szó mögé fűzzék oda az azonosítót is (pl. "Robot13")

4 További játékosok

a) Implementáljunk egy **Mester** osztályt, amelyik a **Jatekos** leszármazottja, és van mester-fokozata (egy egész érték)! A mester-fokozat konstruktorban állítható. A **toString()**-ben a „Mester” és a mesterfokozat térjen vissza. A lépésekkor írja ki a fokozatát, hogy hányadik körnél jár, és minden körben emelje a tétet annyi százalékkal, amennyi a mester-fokozata!

b) Implementáljunk egy **Nyuszi** osztályt, amelyik a **Jatekos** leszármazottja! A színét konstruktorban lehesen állítani, a **toString()**-gel lekérdezni.
A lépésekkor írja ki a színét és hogy hányadik körnél jár! Minden körben emelje a tétet 5-tel addig, amíg a tét 50-nél kisebb. Ha a tét nagyobb lett, mint 50, akkor innentől ne emeljen, csak lépésenként írja ki a tét értékét és azt, hogy „Húha!”.

c) Adjunk az egyik asztalhoz egy mestert és egy nyuszit, játszassunk velük 10 kört!

5 Humán játékos

a) Implementáljunk egy **Ember** osztályt, amelyik a **Jatekos** leszármazottja! Az Ember a lépések során kiírja az aktuális tétet, és a felhasználótól kérdezi, hogy mennyivel emeljen.

b) Adjunk az egyik asztalhoz egy Human-t is, játszassunk velük 10 kört!

II. Geometria

Egyszerű síkidomok modellezése a célunk. Legyen egy síkidom ősosztály, ami megadja, hogy általában mit várhatunk el egy síkidomtól, legyenek konrkét síkidomaink, és ezeket tudjuk együtt kezelni.

6 Alaposztály

Hozzunk létre egy ***Shape*** (alakzat) osztályt, amelynek az alábbi publikus metódusai vannak:

- *double getArea()*: visszadja a területet
- *double getPerimeter()*: visszaadja a kerületet

Van-e értelmes implementációja ezeknek a függvényeknek a Shape osztályban?

- Ha igen, mi az?
- Ha nem, akkor mit kell tenni, hogy ne kelljen megadni implementációt?

7 Leszármazottak

a) Hozzuk létre Shape leszármazottait az alábbi jellemzőkkel:
   - ***Circle*** (kör): r (sugár); kerület: $2r\pi$, terület: $r^2\pi$
   - ***Square*** (négyzet): d (oldalhossz); kerület: $4d$, terület: d^2
   - ***Triangle*** (háromszög): a,b,c (oldahosszak); terület (Herón-képlet): $\sqrt{s(s-a)(s-b)(s-c)}$, kerület: $2s$ (ahol $s = (a+b+c)/2$)
     Szorgalmi feladat: ha az *a,b,c* számhármas nem teljesíti a háromszög-egyenlőtlenséget, akkor a konstruktorból dobjunk *TriangleException*-t!

b) Minden osztálynak legyen *toString* metódusa, ami az osztály nevét és attribútumait írja ki!

8 Összetett alakzat

a) Hozzunk létre egy ***Compound*** nevű osztályt, ami elemi Shape-ekből állhat (pl három négyzetből meg egy körből). Használjunk a tároláshoz ArrayList-et!
b) Lehessen a *Compound*-hoz újabb alakzatot adni (*add*)!
c) El lehessen kérni az *i.* alakzatot (get)! Ez a metódus dobjon *IndexOutOfBoundsException*-t, ha rossz indexet adunk meg! Kell-e ehhez a metódusban ilyen kivételt létrehozni és throw-val eldobni?
d) Implementáljuk a *toString* metódust! Ez menjen végig az egyes tartalmazott *Shape*-eken, és fűzze egybe az egyes *toString*-ek által visszaadott Stringeket!
e) Le lehessen kérdezni, hogy hány eleme van (*size*)!
f) Vegyünk fel egy ilyen objektumot (*comp*) és adjunk hozzá két négyzetet, egy kört és egy háromszöget! Listázzuk ki a tartalmat, és írassuk ki a teljes területet is!
g) Menjünk végig *comp* elemein, és írassuk ki külön-külön a területüket!

9 Rekurzív tárolás

a) Örököltessük a *Compound* osztályt is a *Shape*-ből!
b) A *getArea* adja vissza a tartalmazott alakzatok összterületét!
c) A *getPerimeter* adja vissza a tartalmazott alakzatok kerületének összegét!
d) Hozzunk létre két *Compound* objektumot (*c1*, *c2*), az egyikbe tegyünk 2 db. kört, a másikba 2 db. négyzetet, és *c1*-et és *c2*-t adjuk az előző feladat *comp* objektumához!
e) Listázzuk ki *comp* tartalmát!
f) Írassuk ki *comp* területét!
g) Írassuk ki *comp*-ot String-ként (*toString*)!