Tartalomjegyzék

Bemelegítõ feladatok

Ezeket mind lehet csak egy **main**be írni, egy amúgy üres osztályba. Ha ezt minden file elejére írjátok, akkor nem lesz gond az osztályok importálgatásával:

```
import java.util.*;
```

FONTOS: ahol iterálgatásról meg iterátorról írok, ott megoldható az elõadáson látott **while** ciklusos **hasNext** metódussal és bejáró **for** ciklussal is.

ArrayList

- Készítsetek egy **String**eket tároló **ArrayList**et. Adjátok hozzá a következõ Stringeket:

```
Java
Unix
Oracle
C++
Perl
```

- Majd iterátorral járjátok be a listát és írjátok ki az elemeit külön sorokba (tehát a kimenet legyen az ami pont a fenti felsorolás).
- Töröljétek a C++ és Oracle elemeket. Majd írjátok ki mégegyszer a maradék listát.
- Kérdezzétek le, hogy hanyadik indexen található a Unix. Állítsátok ezt át **Linux**ra. Végül írjátok ki csak ezt a módosított elemet a **get** metódust használva.

HashSet

- **HashSet** segítségével határozzatok meg prímeket.
- Készítsetek egy **int**eket tároló **HashSet**et. Adjátok hozzá a számokat 2-tõl 100-ig. Valamint készítsetek mégegy **HashSet**et, amibe nem töltünk még elemeket.

- Kérjetek el egy iterátort és iteráljatok végig az elemeken. A cikluson belül kérjetek el egy újabb iterátort. Majd az így megírt belsõ cikluson belül vizsgáljátok, hogy a külsõben figyelt elem osztható-e a belsõ elemmel, ha igen és nem egyezik meg ezzel az elemmel, akkor rakjátok a második **HashSet**be.

- Amikor lefutott ez az egymásba ágyazott két ciklus, akkor a második **HashSet**ben megkaptuk az összetett számokat. Töröljétek ezeket az elsõ **HashSet**bõl. (Iteráljatok végig a 2.-on, és hívjátok meg a **remove** metódust az adott elemmel az elsõ **HashSet**en.)

- Gondolkozzatok el rajta hogyan lehetett volna ezt optimálisan megoldani.

HashMap

- Készítsetek egy **HashMap**et, ami Stringekhez számokat tud hozzárendelni. Töltsétek ezt fel a következõ kulcs-érték párokkal:

```
one: 1
two: 2
three: 3
four: 4
five: 5
```

- Írjátok ki csak a kulcsokat, majd csak az értékeket. Végül a kulcs-érték párokat úgy ahogy fent is látszik. Ehhez segítség:

```
Set<String> keys = hm.keySet();
for(String key: keys){
    System.out.println("Value of "+key+" is: "+hm.get(key));
}
```

- hm itt egy **HashMap**.
- Az elsõ sorban lekérjük a kulcsait tartalmazó halmazt.
- Majd egy ciklussal végigiterálunk a kulcsokon.
- A cikluson belül megy a kiírás.

További feladatok

ArrayList különbség

Írjatok egy osztályt (nevet ti találjátok ki), ami egy ArrayList-ben tárolja az elemeket és két ilyen objektumot ki lehet vonni egymásból. Azaz legyen egy metódusa, ami egy ugyanilyen objektumot kap és visszaadja a különbséget. A különbség mûködjön a következõképpen:

- Ha egy elem többször szerepel a **kisebbítend?**ben, de a **kivonandó**ban csak egyszer, akkor csak egy elõfordulással legyen kevesebb a **különbség**ben. Azaz annyival legyen kevesebb a **különbség**ben amennyi a **kivonandó**ban van.
- Ha a **kivonandóban** egy elem többször szerepel, mint a **kisebbítendõben**, akkor töröljük az összes elõfordulását a **különbség**ben, de negatívba ne menjünk (mert nem tudunk).

Elõször talán jobb, ha úgy írjátok meg az osztályt, hogy konkrétan pl egy **ArrayList<String>**-et tárol, de utána próbáljátok megcsinálni általános **ArrayList**-re.

Asteroids powerup

Elõször nézzétek át hogyan is használom az Asteroids-ban az ArrayList-eket az aszteroidák és a golyók tárolására.

Majd találjátok ki, hogyan lehetne powerup-okat a játékba rakni. Elõször egy olyat rakjatok bele, amivel változik a lövésetek. Pl hátrafelé is lõ vagy 3 irányba.

A kirajzoláshoz használhatjátok ezt a kódot:

```
public void draw(Graphics2D g) {
if(alive) {
Graphics2D g2d = (Graphics2D) g.create();
    setColor(Color.BLACK);

    translate(p.getX(), p.getY());

    draw(powerUpDrawable);
    fill(powerUpDrawable);

    dispose();  g2d.
}
}
```

Ahol a konstruktorban hozzátok létre a **powerUpDrawable**-t, a típusa **Polygon**:

```
powerUpDrawable = new Polygon(new int[]{0, 5, 5, 0}, new int[]{0, 0, 5, 5}, 4);
```

Hogy ez mûködjön, kell hogy az objektumnak legyen **Position p** adattagja.

Tippek:

- Collidable-nak kell lennie, és új **ArrayList**-et kell neki nyitni a **MainLoop**-ban.
- Elõször oldjátok meg, hogy ki legyen rajzolva. Ha az megvan akkor már jól álltok. Tudjon utána ütközni, de csak a **SpaceShip**-el.
- Alapból jó, ha csak simán a játék kezdetekor valami adott pozícióban van.
- Ha már tud ütközni, (pl olyankor törlõdik), akkor jöhet a logika, hogy hogyan növeli a lövedékek számát.
- Ehhez kelleni fog egy logikai változó a **SpaceShip**-hez. Ami, ha igaz akkor több lövedék van, ha hamis akkor nem. Ezt állítsa a powerup.
- Majd megoldhatjátok, ha nagyon jól álltok, hogy csak adott ideig mûködjön, aztán álljon vissza az eredeti lövés. (Ehhez kellene egy visszaszámoló változó a **MainLoop**-ba, ami mindig csökken az **actionPerformed**-ban.)
- Majd ehhez lehetne egy új számlálót tenni valahova, ami visszaszámol.
- Stb, stb.