Bálozók

A sikeres Java kurzus elvégzése után hatalmas bált szerveznek. Ezen lesz bálkirálynő választás is. Írjon Java programot a bál szimulálására!

Minden egyes **bálozó**t egyértelműen jellemez a *neve* és a belépéskor kapott *sorszáma*. Belépéskor mindenkinek van *zsebpénz*e. Ebből tudnak majd fizetni, de kívülről már nem kaphatnak több pénzt.

Az est folyamán mindegyikük *táncol()*, mindenki *fogyaszt()* (ha tud), és mindenki *szavaz()*hat egy **lány**ra. (Egyszerre egyre, de elvileg akárhányszor szavazhat.) Minden egyes táncoláskor eggyel növekszik a *táncai száma*. Szavazáskor a paraméterben adott lány szavazatot kap. Fogyasztani pedig nyilván csak akkor tud, ha még ki tudja fizetni a metódus paraméterében szereplő árat.

A lányok *szavazatot kap()*hatnak. Ekkor *szavazataik száma* eggyel nő.

Kiíratáskor majd azt kell megadni, hogy az illető fiú vagy lány, a nevét, sorszámát, és azt, hogy hányszor táncolt.

A **vezérlő** osztályban a szükséges adatok beállítása után *léptesse be()* a bálozókat – olvassa be az adataikat egy adatfájlból (pl. balozok.txt – adatszerkezet: név;fiú/lány;zsebpénz).

Ezután *bálozzanak()* egészen hajnalig (ezt most úgy szimulálja, hogy a véletlen dönti el, hogy vége van-e vagy nincs). Eközben egy-egy véletlenül kiválasztott bálozó fizessen véletlen értékben, feltéve persze, hogy tud még fizetni. Ha nem, akkor jelezze. Egy másik véletlenül kiválasztott bálozó próbáljon meg szavazni egy véletlenül kiválasztott emberre. Ha lányt választott, akkor szavazzon rá, ha történetesen fiút, akkor kapjon egy rosszalló hibaüzenetet. Egy harmadik véletlenül választott szavazó pedig táncoljon. (Programozási szempontból lényegtelen, de ha akarja, választhat egynél több táncost is.)

Végül összesítse a bál tapasztalatait: írassa ki, hogy ki mennyit költött, illetve melyik lány hány szavazatot kapott.

Ki(k) lett(ek) a bálkirálynő(k)?

Mennyi volt a büfé összes bevétele? (Összes költekezés.)

Írassa ki a bál résztvevőit

a) táncok száma szerint csökkenően

b) fogyasztás szerint növekvően

c) szavazatszám szerint csökkenően rendezve.

Hölgyválasz: vagyis el lehessen dönteni, hogy szerepel-e a bálozók között egy adott fiú.

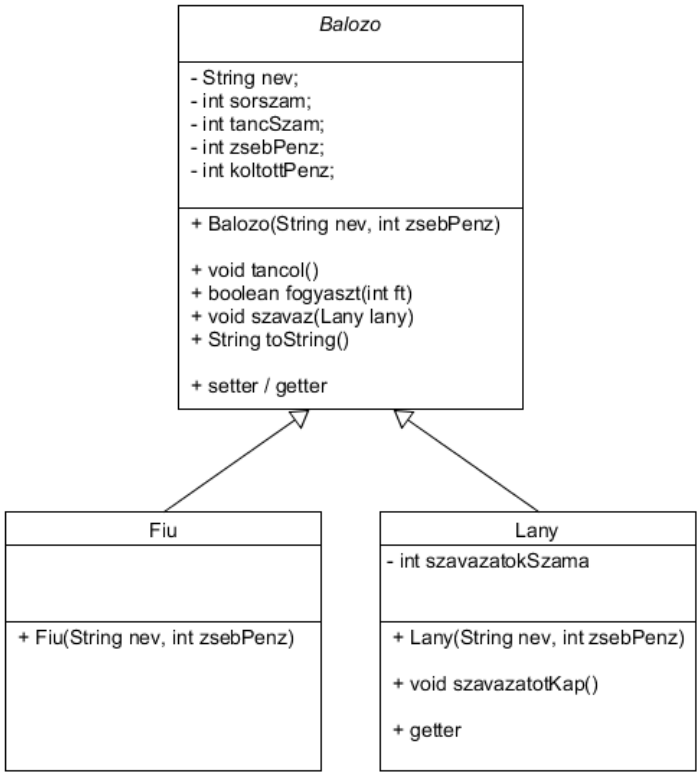
Megoldási javaslat:

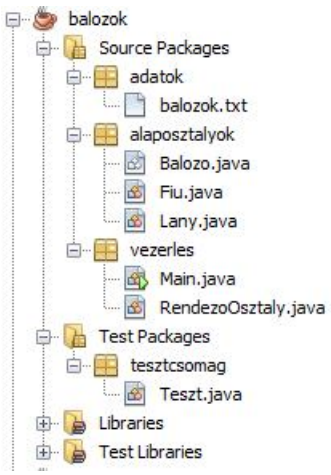
A feladat szerint meg kell különböztetnünk a lányokat és fiúkat, mert bár nagyon sok közös tulajdonságuk van, de szavazni csak lányokra lehet. Emiatt célszerű a közös tulajdonságokat, metódusokat egy közös ősosztályban megfogalmazni, de mivel nem lesz ilyen típusú példány, ezért deklarálhatjuk absztrakt osztályként is.

Egy fiúnak gyakorlatilag nincs is más tulajdonsága, mint az ősosztálynak, ezért elvileg úgy is meg lehetne oldani a feladatot, hogy csak `Balozo` és `Lany` osztály van, de ez zavaróan logikátlanná és nehezen olvashatóvá tenné a kódot.

A bálozó valamekkora zsebpénzzel rendelkezik, ebből tud költeni, de azt is számon kell tartanunk, hogy mennyi pénzt költött, hiszen ez mutatja a fogyasztását.

Az osztályszerkezet a baloldali ábrán látható UML diagram segítségével írható le, a csomagszerkezet a jobboldali ábrán látható:





Lássuk az egyes osztályokat!

`Balozo` **osztály:**

Nincs benne túl sok új ismeret. Az egyedi sorszámra vonatkozó megoldást már megbeszéltük a 2. állatverseny feladat megoldásában (*allatverseny2.pdf*). A másik – de szintén egyszerű – kérdés a fogyasztás. Bár ha már ezeket a feladatokat csinálja, akkor nyilván túl van a programozási alapokon, de talán mégsem árt néhány szót ejteni a metódusok típusáról. Nem

mindig egyértelmű, hogy milyen típusú legyen a megírandó metódus, bár egy kicsit gyakorlattal ez nem okozhat problémát. Nagyon sokat segít a döntésben a feladat elemző végiggondolása. Ha egy értéket kell kiszámolni, ráadásul mindig ugyanúgy, akkor az erre vonatkozó metódus `int` vagy `double`, esetleg `float` típusú. Ha valamilyen cselekvést kell leírni, akkor `void`. Nos, a fogyasztás cselekvés, a feladat szövegében is a „fogyaszt” ige szerepel, vagyis azt gondolnánk, hogy `void` metódust kell írnunk. Ez első pillantásra igaz is lehet, csak ha jobban belegondolunk, ez a cselekvés vagy sikerül, vagy nem, hiszen kérdéses, hogy van-e elég pénzünk. Ha azt is szeretnénk tudni, hogy sikeres-e a művelet, akkor `void` helyett gyakran `boolean` metódust célszerű írni. Sőt, olyan is előfordulhat, hogy egy cselekvés nem csak két-, hanem többféle módon végződhet. Ekkor esetleg `int` típusú metódus ír le egy cselekvést is, ami a cselekvés kimenetelétől függően más-más értéket ad vissza.

Az osztály setterek, getterek és `toString()` nélküli kódja:

```
public abstract class Balozo {

    private String nev;
    private int sorszam;
    private int tancSzam;
    private int zsebPenz;
    private int koltottPenz;

    private static int utolsoSorszam;

    public Balozo(String nev, int zsebPenz) {
        this.nev = nev;
        this.zsebPenz = zsebPenz;
        Balozo.utolsoSorszam++;
        this.sorszam = Balozo.utolsoSorszam;
    }

    /** Táncolás során mindig eggyel növekszik a táncok száma ...3 lines */
    public void tancol() {
        tancSzam++;
    }

    /** A fizetendő ártól függ, hogy tudunk-e fogyasztani ...8 lines */
    public boolean fogyaszt(int ar) {
        if (koltottPenz + ar < zsebPenz) {
            koltottPenz += ar;
            zsebPenz -= ar;
            return true;
        }
        return false;
    }

    /** A paraméterben szereplő lányra szavaz ...5 lines */
    public void szavaz(Lany lany) {
        lany.szavazatotKap();
    }
```

A hiányzó részeket nyilván önállóan is meg tudja írni, de azért gondoljuk végig a setterek, getterek kérdését.

Setterek: Egyetlen setter sem kell, de nem ugyanazon ok miatt. A `nev` változóhoz elvileg írhatnánk, de ha mereven vesszük a megoldást, akkor nem, hiszen egy bál során senki sem változtat nevet. A `zsebpenz` mezőről a feladat előírja, hogy ha valaki belépett a bálba, akkor kívülről már nem kaphat értéket. A `tancSzam`, `koltottPenz` változók értéke egy-egy metódusban változik, ezért nem szabad settert írni hozzájuk. És ugyancsak nem szabad az `utolsoSorszam` és `sorszam` változókhoz, hiszen ezzel épp az egyediségüket veszítenék el.

Getterek: Elvileg minden mezőhöz rendelhetünk gettert, ténylegesen azonban logikus lenne, ha a `zsebpenz` változóhoz nem rendelnénk, hiszen általában nem nagyon szeretjük, ha publikus az, hogy mennyi pénzünk van.

`Fiu` osztály:

Gyakorlatilag ugyanaz, mint a `Balozo` osztály:

```
public class Fiu extends Balozo{

    public Fiu(String nev, int zsebPenz) {
        super(nev, zsebPenz);
    }
}
```

Egy esetben azonban lehet eltérés. Térjünk vissza egy kicsit a `toString()` kérdésére. A feladat szerint azt is vissza kellene adni, hogy fiú vagy lány az illető. Vagyis az ősosztály metódusa:

```
    @Override
    public String toString() {
        return "A " + this.getClass().getSimpleName().toLowerCase()
                    + " sorszáma: " + sorszam + ", neve: " + nev;
    }
```

Csakhogy ezzel van egy kis szépséghiba, hiszen kiíratáskor a „fiu” vagy „lany” szöveget látjuk, ékezet nélkül. Ha még a helyesírásra is oda szeretnénk figyelni – márpedig ez nagyon fontos, hiszen a felhasználó a felülettel találkozik, és ha igényes, akkor visszatetszést kelt benne a sok helyesírási hiba (bár talán az ékezet nélküliség nem annyira). Ha a helyesírásra is oda akarunk figyelni, akkor ez például megoldás lehet:

Az ősosztályban deklarálunk egy absztrakt metódust, amelyet majd az utódok felülírnak:

```
    abstract public String osztalyNev();
```

Ha az ősosztályt nem absztraktra deklaráltuk, akkor ez a metódus adja vissza az üres stringet.

A `toString()` metódusban értelemszerűen ezt a metódust hívjuk meg az osztálynév meghatározására szolgáló metódus helyett.

A `Fiu` osztályban:

```
@Override
public String osztalyNev() {
    return "fiú";
}
```

És nyilván hasonló a `Lany` osztályban is, ott már nem jelezzük.

Megjegyzés: Ennél elegánsabb megoldás az, ha a programban szereplő egyetlen, kiíratásra váró, string értéket sem „égetünk be", hanem egy külön táblázatban tároljuk őket, amely segítségével könnyedén válthatjuk majd a kiíratás nyelvét.

A `Lany` osztály (getter és a már megbeszélt metódus nélkül):

```
public class Lany extends Balozo{

    private int szavazatokSzama;

    public Lany(String nev, int zsebPenz) {
        super(nev, zsebPenz);
    }

    public void szavazatotKap(){
        szavazatokSzama ++;
    }
```

A továbblépés előtt önálló feladatként tesztelje az elkészült alaposztályokat!

A vezérlésből már sok mindenről volt szó korábban, ezeket a kódrészleteket magyarázat nélkül közöljük:

```
public class Main {

    private String CHAR_SET = "UTF-8";
    private final String ADATFAJL = "/adatok/balozok.txt";

    private int FIZETES_HATAR = 1000;
    private double BAL_VEGE_ESELY = 0.1;

    private List<Balozo> balozok = new ArrayList<>();

    public static void main(String[] args) {
        new Main().start();
    }
```

```java
private void start() {
    belep();
    balozas();
    eredmenyHirdetes();
    balkiralynok();
    osszBevetel();
    tancokSorrendje();
    fogyasztasiSorrend();
    szavazatokSzerintiSorrend();
    holgyvalasz();
}

/**
 * A fájlban lévő adatok beolvasása. Adatszerkezet, pl.:
 * Kata;lány;2000
 * Béla;fiú;3000
 */
private void belep() {
    try(InputStream ins = this.getClass().getResourceAsStream(ADATFAJL);
        Scanner sc = new Scanner(ins, CHAR_SET)) {

        String[] adatok;
        String sor;

        while (sc.hasNextLine()) {
            sor = sc.nextLine();
            try {
                if (!sor.isEmpty()) {
                    adatok = sor.split(";");
                    switch (adatok[1]) {
                        case "lány":
                            balozok.add(new Lany(adatok[0],
                                    Integer.parseInt(adatok[2])));
                            break;
                        case "fiú":
                            balozok.add(new Fiu(adatok[0],
                                    Integer.parseInt(adatok[2])));
                            break;
                        default:
                            throw new Exception();
                    }
                }
            } catch (Exception e) {
                System.out.println("Hibás a " + sor + " adatsor.");
            }
        }
    } catch (Exception ex) {
        System.out.println("Hibás fájlmegadás");
        System.exit(-1);
    }
}
```

```
private void balozas() {
    int fogyasztIndex, szavazIndex, kireIndex, tancolindex, penz;
    Lany lany;
    Balozo balozo;
    boolean vege = false;

    System.out.println("\nBál-szimuláció:");
    do {
        fogyasztIndex = (int) (Math.random() * balozok.size());
        szavazIndex = (int) (Math.random() * balozok.size());
        kireIndex = (int) (Math.random() * balozok.size());
        tancolindex = (int) (Math.random() * balozok.size());

        System.out.println("Szavazás:");
        if (balozok.get(kireIndex) instanceof Lany) {
            lany = (Lany) balozok.get(kireIndex);
            balozok.get(szavazIndex).szavaz(lany);
            System.out.println(lany.getNev() + " szavazatot kapott");
        } else {
            System.out.printf("Na ne viccelj, %s fiú\n",
                    balozok.get(kireIndex).getNev());
        }

        balozok.get(tancolindex).tancol();
        System.out.println(balozok.get(tancolindex).getNev() + " táncol");

        System.out.println("\nFogyasztás:");
        penz = (int) (Math.random() * FIZETES_HATAR);
        balozo = balozok.get(fogyasztIndex);
        if (balozo.fogyaszt(penz)) {
            System.out.println(balozo.getNev() + " fogyasztott");
        } else {
            System.out.println(balozo.getNev() + "nak nincs elég pénze");
        }

        if (Math.random() < BAL_VEGE_ESELY) vege = true;

    } while (!vege);
    System.out.println("\nVége a bálnak.");
}

private void osszBevetel() {
    int ossz = 0;
    for (Balozo balozo : balozok) {
        ossz += balozo.getKoltottPenz();
    }

    System.out.println("\nA büfé bevétele: " + ossz + " Ft");
}
```

```
private void eredmenyHirdetes() {
    System.out.println("\nA bálozók fogyasztása: ");
    for (Balozo balozo : balozok) {
        System.out.printf("%s, %d Ft-t költöt\n",
                balozo, balozo.getKoltottPenz());
    }
    System.out.println("\nA lányok szavazatai");
    for (Balozo balozo : balozok) {
        if (balozo instanceof Lany) {
            System.out.printf("%s, %d szavazatot kapott\n",
                    balozo, ((Lany) balozo).getSzavazatokSzama());
        }
    }
}

private void balkiralynok() {
    int max = 0;
    for (Balozo balozo : balozok) {
        if (balozo instanceof Lany) {
            max = ((Lany) balozo).getSzavazatokSzama();
            break;
        }
    }
    for (Balozo balozo : balozok) {
        if (balozo instanceof Lany
                && ((Lany) balozo).getSzavazatokSzama() > max) {
            max = ((Lany) balozo).getSzavazatokSzama();
        }
    }

    System.out.println("\nA legnépszerűbb lány(ok) " + max
                                                    + " szavazattal:");
    for (Balozo balozo : balozok) {
        if (balozo instanceof Lany
                && ((Lany) balozo).getSzavazatokSzama() == max) {
            System.out.println(balozo.getNev());
        }
    }
}
```

Megjegyzések:

1. A balkiralynok() metódusnak van egy kis hibája, ugyanis nem kezeli azt az – egyébként szokatlan – esetet, ha a bálban egyáltalán nincsenek lányok. Majd később, valamelyik feladat kapcsán ezt a megoldást is megbeszéljük, most esetleg próbálkozzon vele önállóan.

2. Felmerülhet a kérdés, hogy mi és mennyi legyen egy-egy metódus tartalma. Fontos elv a modularitás, vagyis az, hogy minden metódus egyetlen funkciót valósítson meg. Ezt tudva kérdés, hogy vajon jó-e a `balozas()` metódus, hiszen abban – legalábbis látszólag – több funkció is szerepel. Ez igaz, csakhogy ezek a részletek nem választhatóak el egymástól,

hiszen ha három külön ciklust írnánk, akkor már három különböző bálról lenne szó. Vagyis ezek összetartoznak, és a látszat ellenére egyetlen funkciót, a bálozást valósítják meg. Az már más kérdés, hogy lehet másképp is szervezni, például így:

```
private void balozas() {

    boolean vege = false;
    System.out.println("\nBál-szimuláció:");
    do {
        szavazas();
        tancolas();
        fogyasztas();

        if (Math.random() < BAL_VEGE_ESELY) {
            vege = true;
        }

    } while (!vege);
    System.out.println("\nVége a bálnak.");
}
```

ahol értelemszerűen a hivatkozott metódusok oldják meg a korábbi részleteket. Hogy kinek melyik megoldás a szimpatikusabb, ízlés kérdése. Az általában célszerű, hogy egy metódus lehetőleg ne legyen hosszabb egy képernyőképnél.

Végül beszéljük meg a rendezést és a kiválasztást.

Az 1. állatverseny feladat megoldásában (*allatverseny1.pdf*) már volt szó rendezésről. Megbeszéltük, hogy ahhoz, hogy rendezni tudjuk az objektumokat, meg kell adnunk a rendezési szempontot. Ezt megtehetjük úgy, hogy a rendezendő objektumokat leíró osztályt `Comparable` típusúként deklaráljuk (`implements Comparable<>`) és implementáljuk a hozzá tartozó `compareTo()` metódust, illetve úgy is, hogy egy külön, `Comparator` típusú, rendező osztályt definiálunk (`extends Comparator<>`), és implementáljuk a hozzá tartozó `compare()` metódust. Ha az egyik változatot érti, akkor érti a másikat is ☺, ezért most csak az egyiket beszéljük meg, mégpedig a külön rendező osztály, ekkor ugyanis nem kell hozzányúlni a korábban megírt `Balozo` osztályhoz.

A korábbiakhoz képest annyi az újdonság, hogy többféle szempont szerinti rendezést is lehetővé szeretnénk tenni. Lássuk az osztály kódját, majd utána egy kis magyarázatot:

```
public class RendezoOsztaly implements Comparator<Balozo>{

    public enum Szempont{TANCOK_SZAMA, FOGYASZTAS, SZAVAZATSZAM}

    public static final boolean NOVEKVOEN = true;
    public static final boolean CSOKKENOEN = false;

    private static Szempont valasztottSzempont;
    private static boolean miModon;

    @Override
    public int compare(Balozo o1, Balozo o2) {
        switch (valasztottSzempont) {
            case TANCOK_SZAMA:
                return miModon ? o1.getTancSzam() - o2.getTancSzam()
                        : o2.getTancSzam() - o1.getTancSzam();
            case FOGYASZTAS:
                return miModon ? o1.getKoltottPenz() - o2.getKoltottPenz()
                        : o2.getKoltottPenz() - o1.getKoltottPenz();
            case SZAVAZATSZAM: {
                int o1SzavazatSzam, o2SzavazatSzam;
                o1SzavazatSzam = (o1 instanceof Lany)
                        ? ((Lany) o1).getSzavazatokSzama() : 0;
                o2SzavazatSzam = (o2 instanceof Lany)
                        ? ((Lany) o2).getSzavazatokSzama() : 0;
                return miModon ? o1SzavazatSzam - o2SzavazatSzam
                        : o2SzavazatSzam - o1SzavazatSzam;
            }
            default:
                return 0;
        }
    }

    public static void setValasztottSzempont(Szempont valasztottSzempont,
                                                boolean miModon) {
        RendezoOsztaly.valasztottSzempont = valasztottSzempont;
        RendezoOsztaly.miModon = miModon;
    }

    public static Szempont getValasztottSzempont() {
        return valasztottSzempont;
    }

    public static boolean isMiModon() {
        return miModon;
    }
}
```

Látható, hogy a `compare()` metódusban egy `switch-case` szerkezet gondoskodik arról, hogy más-más szempont szerint tudjunk rendezni. A szempontokat célszerű `enum`-ként megadni, mert így kisebb a tévesztés esélye. A `miModon` változóban azt tudjuk beállítani, hogy növekvően vagy csökkenően szeretnénk rendezni. Nyilván elég lenne `true` vagy `false` értékeket megadni, de jóval olvashatóbb, és a felhasználó számára is kényelmesebb, ha nem

igaz vagy hamis értéket kell megadnia, hanem közölheti, hogy növekvően vagy csökkenően szeretne rendezni. Emiatt vezettük be a `NOVEKVOEN`, `CSOKKENOEN` konstansokat.

Még egy apróság: mivel a szempontot és a rendezés módját együtt célszerű megadni, ezért kényelmesebb két setter helyett egyetlen olyan metódust írni, amely mindkét változó értékét beállítja.

A vezérlő osztály rendező metódusai:

```
private void tancokSorrendje() {
    RendezoOsztaly.setValasztottSzempont(RendezoOsztaly.Szempont.TANCOK_SZAMA,
            RendezoOsztaly.CSOKKENOEN);
    Collections.sort(balozok, new RendezoOsztaly());
    balozokKiirasa("\nA táncok száma szerint rendezve: ");
}

private void fogyasztasiSorrend() {
    RendezoOsztaly.setValasztottSzempont(RendezoOsztaly.Szempont.FOGYASZTAS,
            RendezoOsztaly.NOVEKVOEN);
    Collections.sort(balozok, new RendezoOsztaly());
    balozokKiirasa("\nA fogyasztás szerint rendezve: ");

}

private void szavazatokSzerintiSorrend() {
    RendezoOsztaly.setValasztottSzempont(RendezoOsztaly.Szempont.SZAVAZATSZAM,
            RendezoOsztaly.CSOKKENOEN);
    Collections.sort(balozok, new RendezoOsztaly());
    balozokKiirasa("\nA szavazatok száma szerint rendezve: ");
}

private void balozokKiirasa(String cim) {
    String tempFormat = "%s, %d tánc, %d Ft fogyasztás";
    System.out.println(cim);
    for (Balozo balozo : balozok) {
        System.out.printf(tempFormat,
                balozo, balozo.getTancSzam(), balozo.getKoltottPenz());
        if (balozo instanceof Lany) {
            System.out.println(", " + ((Lany) balozo).getSzavazatokSzama()
                                                    + " szavazat");
        } else {
            System.out.println();
        }
    }
}
```

Természetesen lehetne még szépíteni a kiíratást, de talán a célnak ez is megfelel.

Beszéljük meg a hölgyválaszt, vagyis azt, hogyan lehet eldönteni, hogy egy adott nevű fiú szerepel-e a bálozók között. Azonnal megjegyezzük, hogy a név sohasem azonosítja egyértelműen az illetőt, vagyis most csak azt oldjuk meg, hogy szerepel-e ilyen nevű fiú. Az már önálló feldolgozás lehet, hogy pl. írassa ki az összes ilyen nevű fiút – ha egyáltalán talál ilyet.

Az lehet az első ötlete, hogy ez egy viszonylag egyszerű feladat: ha ismerjük (beolvastuk) a keresett nevet, akkor egy ciklussal végigmegyünk a bálozók listáján, és ellenőrizzük, hogy az aktuális bálozó neve azonos-e a megadott névvel. Ha csak a fiúk között akarjuk keresni, akkor előbb még azt is ellenőrizni kell, hogy az illető fiú-e. Nem tűnik túl bonyolultnak, és nem is az. Működne az ötlet. De talán még egyszerűbb ez: létrehozunk egy ilyen nevű `Fiu` típusú példányt, és ellenőrizzük, hogy a bálozók listája tartalmazza-e. Vagyis:

```
Fiu fiu = new Fiu(nev, 0);
if(balozok.contains(fiu)) System.out.println("Szerepel");
else System.out.println("Nem szerepel");
```

És hogy ne kelljen minden egyes keresés kedvéért újrafuttatni a programot, az egészet belerakjuk egy ciklusba:

```
private void holgyvalasz() {
    Scanner scanner = new Scanner(System.in);
    String nev;
    do {
        System.out.print("Kit választ? ");
        nev = scanner.nextLine();
        Fiu fiu = new Fiu(nev, 0);
        if(balozok.contains(fiu)) System.out.println("Szerepel");
        else System.out.println("Nem szerepel");

        System.out.println("Keres még valakit? (i/n)");
    } while ("i".equals(scanner.next()));
}
```

(Azt már oldja meg önállóan, hogy a keresés folytatására vonatkozó kérdésre valóban csak i/n választ lehessen adni.)

Ez nagyon kényelmes megoldásnak tűnik, csak sajnos nem jó – illetve szerencsére csak még nem jó, de javítjuk. A baj az, hogy a frissen létrehozott `Fiu` példány valóban nem szerepel a korábban létrehozott listában. No de nem is azt szeretnénk tudni, hogy ő szerepel-e, hanem csak azt, hogy egy ilyen szerepel-e. Márpedig azt meg tudjuk mondani, hogy mikor tekintünk egyformának két különböző példányt. Ezt az osztály `equals()` metódusában definiálhatjuk. Emellett a `hashcode()` metódust is meg kell írni, ami az objektum hash kódját állítja elő. Ezeket a fejlesztőkörnyezet tudja generálni, csak azt kell megadnunk, hogy mi alapján tekintünk egyformának két objektumot. Esetünkben ez a név.

Mivel a `Fiu` egy utódosztály, ezért nem itt, hanem az ősosztályban definiáljuk ezeket a metódusokat:

```java
@Override
public int hashCode() {
    int hash = 5;
    hash = 23 * hash + Objects.hashCode(this.nev);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Balozo other = (Balozo) obj;
    if (!Objects.equals(this.nev, other.nev)) {
        return false;
    }
    return true;
}
```

Ha „helyesen” olvasta ezt a leírást, vagyis úgy, hogy közben saját maga is csinálta, akkor csaknem biztos, hogy más hash érték adódott a generálás során. De ez természetes, hiszen nem ugyanazon a gépen dolgozunk, sőt, ugyanazon a gépen dolgozva is más-más eredmény adódik, ha ismét generálja.

Végére jutottunk a megoldásnak ☺