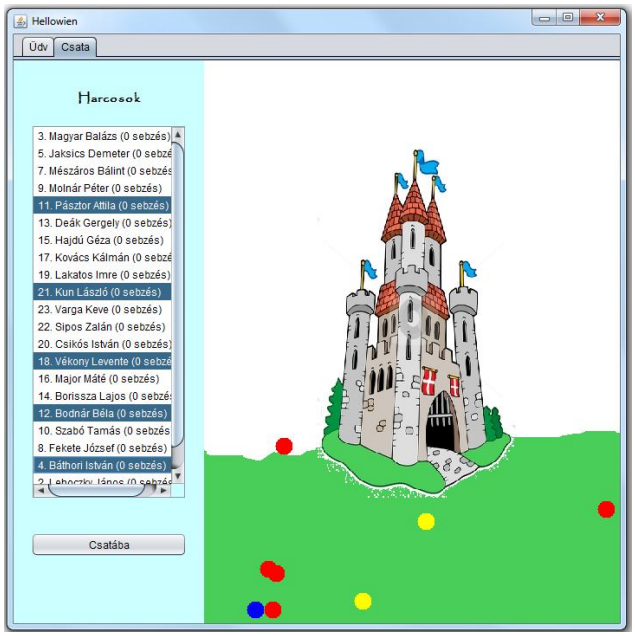
Feladat:
Egyre trendibb megünnepelni a Hellowien-t, nézzük meg, hogy mit is ünneplünk ekkor, és írjunk hozzá egy programot. A program induló felülete:



Kétségtelen, hogy a dátum egy kicsit hibás, de ekkora távlatból egy kis kerekítési hiba elmegy, a lényeg akkor is az, hogy „S nyögte Mátyás bús hadát Bécsnek büszke vára.”

Szóval, az első tabulátor-felületen látható az üdvözlő oldal, a másodikon a csatát „szimulál-juk”. Most csak egy-egy pötty jelzi a harcosokat, de ezt később lehet módosítani.
Ez a felület további két részből áll. A baloldalon a harcosok névsora látható, a jobboldalon a csatatér. Méretek: A teljes panel-felület: 700*650, a harcosok listáját tartalmazó rész 220 pixel szélességű.

A harcosok adatai egy Derby adatbázisba kerülnek.
A tábla neve: KATONAK.

id	nev	rang
1	Kinizsi Pál	lovas parancsnok
2	Lehoczky János	gyalogos parancsnok
3	Magyar Balázs	gyalogos parancsnok
4	Báthori István	lovas parancsnok
5	Jaksics Demeter	lovas parancsnok
6	Both János	lovas parancsnok
7	Mészáros Bálint	lovas
8	Fekete József	lovas
9	Molnár Péter	gyalogos
10	Szabó Tamás	gyalogos
11	Pásztor Attila	lovas
12	Bodnár Béla	lovas
13	Deák Gergely	gyalogos
14	Borissza Lajos	gyalogos
15	Hajdú Géza	lovas
16	Major Máté	gyalogos
17	Kovács Kálmán	gyalogos
18	Vékony Levente	gyalogos
19	Lakatos Imre	gyalogos
20	Csikós István	lovas
21	Kun László	zászlós
22	Sipos Zalán	zászlós
23	Varga Keve	zászlós

Az adatbázisból létrehozott példányok egy rendezhető lista-modellbe kerülnek – a rendezési szempont a katonák sebzési értéke. Mint a fenti ábrán látható, mindegyik katonának van egy egyedi sorszáma is.

A rangtól függően más-más színű, de egyforma méretű pöttyök jellemzik őket. (A képen látható megoldásban a lovas parancsnok kék, a lovas piros, a gyalogos parancsnok zöld, a gyalogos sárga, bárki más fekete, de bármilyen más színt is kitalálhat.) Később esetleg sok minden másban is eltérhetnek egymástól.

A katonákat a nevük, egyedi sorszámuk definiálja. Mindegyik lőtt, és mindegyiket meglőtték (persze, ha szerencséje van, akkor nem, de elvileg mindkettő lehet). Ha lőtt, akkor a sebzés értéke eggyel nő, ha meglőtték, akkor eggyel csökken, és ha negatívvá válik, akkor szegény katona meghal. Csak élő katona lőtt, ill. őt lehet meglőni.

A „Csatába” feliratú gomb hatására a listából kijelölt katonák harcba indulnak. Ez azt jelenti, hogy a csatatéren véletlen helyeken megjelennek az őket jellemző pöttyök. (Mivel akkor még

nem volt légierő, ezért lehetőleg ne a levegőben jelenjenek meg, hanem – a háttérképtől függő magasságig, mondjuk, a panel fele (vagy valahány százalékának) magasságáig.)
A gomb megnyomásakor szűnjön meg a kijelölés. (A gomb újbóli megnyomásakor a régi harcosok eltűnnek, újak kerülnek a csatatérre.)

Az a katona lőtt (vagy lövik meg), akire az egérrel rákattintunk. Ekkor valahány százalék eséllyel meglövik vagy ő lő. (Ezt a véletlen dönti el.)
Természetesen ennek megfelelően állandóan változik majd a listában látható sorrend (sebzések szerint csökkenően), illetve ha egy katona meghal, akkor a listából is, csatatérről is töröljük.

A megoldás során ezeket a fogalmakat (is) használjuk: adatbázisból való olvasás, singleton tervezési minta, gyártó függvény, rajzolás, leegyszerűsített MVC (ill. MVP) modell.

Egy lehetséges megoldás vázlata:

1. Megcsináljuk az üdvözlő oldalt – szerintem ezt mindenki meg tudja oldani önállóan is. (Ha nem megy, használja a képrajzoláshoz kiadott kis segédfájlt.)

2. A második felület kialakítása – ezt sem nehéz önállóan megoldani, de néhány támpont: Kétféle módon is kialakíthatjuk:
   a) Egyetlen panelre kerül a listafelület is és a „csatatér” is. Ez esetünkben elég egyszerű és kézenfekvő megoldás lehet, csak a későbbiekben arra kell figyelni, hogy a háttérkép nem a panel szélén, hanem beljebb kezdődik, illetve a harcosokat szimbolizáló pöttyök helye sem a panel szélén, hanem beljebb kezdődhet. A konkrét feladat szempontjából ez lenne az egyszerűbb megoldás, nyugodtan próbálkozhat vele.
   b) A másik lehetőség, hogy két külön panelen kezeljük a listafelületet és a csatateret. Ez a konkrét esetben most inkább elbonyolítást jelent, de több oka van annak, hogy mégis ezt a megoldást választottuk. Az egyik (de most kevésbé lényeges szempont): így elválasztható egymástól a listafelülettel kapcsolatos rész és a csatajelenet, ez egy esetleges továbbfejlesztéskor lehet érdekes. A lényegesebb szempont az volt, hogy ennek kapcsán arról is szó eshetett, hogy hogyan lehet kapcsolatot teremteni két panel között. (A két panel: a harcosok listáját tartalmazó a `HarcosokPanel`, a csatatér a `CsataPanel`-en van.)

3. Megoldjuk az adatbevitelt. Ehhez több dolgot kell végiggondolnunk:
   - Hogy épüljön fel a Katona osztály?
   - Hogyan oldjuk meg a rangok kezelését?
   - Hogyan olvassunk be a megadott adatbázisból?
   - Hol hívjuk meg a beolvasást?

4. Oldjuk meg a „csatajeleneteket”, vagyis azt, hogy a gombnyomás hatására megjelennek a harcosok a csatatéren, a kiválasztott katona harcol, és harcának eredménye megjelenik a listafelületen (beleértve a halott katonák eltávolítását is).

Haladjunk a fenti vázlatpontok alapján.

Osztályok és adatbevitel

Katona osztály:
Funkciók szempontjából három részre tagolható:
- Viselkedésével kapcsolatos metódusok (lő, meglőtték, él-e, sebzési érték).
- Adminisztrációhoz szükséges dolgok (név, sorszám, `toString()`).
- Geometriai ábrázolásához szükséges adatok (középpont, sugár, szín, ill. a kirajzolás).

Rangok kezelése:
Ha csak nagyon szűken, a konkrét esetre koncentrálnánk, akkor a rangot akár string-ként is kezelhetnénk, és valamilyen elágazás (`switch-case`) segítségével beállíthatnánk, hogy melyik ranghoz milyen szín tartozik. Ez azonban nagyon leszűkítené a megoldást, és nem ad teret a későbbi bővítésre. Ha egy esetleges továbbfejlesztésre is gondolunk, akkor célszerűbb annyi utódosztályt létrehozni, ahányféle rang van. Ekkor megoldható az, hogy pl. a zászlósokat ne pöttyel, hanem zászlóval jelezzük, vagy, hogy pl. más módon sebez egy lovas, mint egy gyalogos, stb.
Most csak annyi szerepel az utódosztályokban, hogy a példány létrehozásakor beállítjuk az adott ranghoz tartozó színt.

Az adatbeolvasást megvalósító osztály:
Itt két dolgot kell végiggondolnunk:
1. Adatbázishoz való kapcsolódás és az adatok lekérése.
2. A beolvasott rang alapján hogyan lehet létrehozni a megfelelő típusú katona-példányt.

Az adatbázisból való olvasás már nem újdonság, ezek a lépései:
- kapcsolódunk az adatbázishoz,
- ha létrejött a kapcsolat, akkor lekérünk tőle egy utasításobjektumot, és ennek segítségével közöljük az adatbázissal a végrehajtandó SQL utasítást
- a kapott eredményhalmazt feldolgozzuk, és az adatok alapján létrehozzuk a megfelelő példányokat, és hozzáadjuk a kialakítandó konténerhez (esetünkben egy rendezhető listamodellhez, bár létezik más jó megoldás is.).

A katona-példányok létrehozásának nagyon fapados és nem túl elegáns módja az, hogy megírunk egy `switch-case` szerkezetet, amely közli, hogy milyen ranghoz milyen utód-típus tartozik. Ez azért nem szerencsés megoldás, mert nagyon bedrótozza a rangokat, nehéz bővíteni egy esetleges újabb ranggal, ráadásul a sok `break` még nehézkesebbé is teszi az egészet.
Sokkal szebb megoldás a gyártófüggvény (*factory method*) tervezési minta használata. Ennek lényege: létrehozunk egy „gyárat”, mondjuk `KatonaGyar` néven, vagyis egy olyan osztályt, amelynek van egy gyártó függvénye (gyártó metódusa). Ez a megadott rang és név alapján létrehozza a szükséges típusú katona-példányt.
Mivel a gyár osztályból elég egyetlen példány, ezért a *singleton* tervezési mintát is kipróbáljuk.

A beolvasás meghívása:
Ha megírtuk a korábban megbeszélt osztályokat, akkor már csak az a kérdés, hogy hol hívjuk meg ezt a beolvasást.

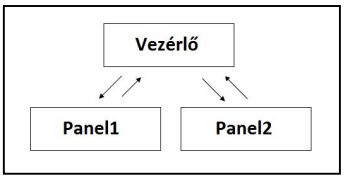
Mondhatjuk azt, hogy mivel a `HarcosokPanel` tartalmazza a listafelületet, ennek betöltéséhez kapcsoljuk a beolvasást. Ez jó is lenne, ha nem tabulált felületen lennénk, és csak ez, a két panelt tartalmazó felület lenne az egyetlen. Most azonban az a baj, hogy a regiszter-fülek váltakozásával újra- és újra betöltődik a panel, vagyis ismét és ismét meghívja a beolvasást, és létrehoz újabb és újabb katonákat.

Mivel a két panel közötti kapcsolat koordinálásához majd egy vezérlő osztályt használunk, ezért logikus, ha mindent erre a vezérlőre bízunk, és itt hívjuk meg a beolvasást is. Ekkor a frame-nek összesen annyi a dolga, hogy elindítsa a vezérlő `start()` metódusát. Így a frame is függetleníthető a vezérlőtől.

A harcosok listáját és a csatamezőt tartalmazó felület és a hozzá tartozó logika

Mint megbeszéltük, most azt a változatot választjuk, amely két külön panelen kezeli a listafelületet és a csatamezőt.
A kérdés az, hogyan teremthetünk kapcsolatot a két panel között.
A földhözragadt megoldás az, hogy az egyik panel közvetlenül üzen a másiknak (azaz közvetlenül meghívja annak megfelelő metódusait), és persze, viszont. Ez persze akár meg is oldhatja az aktuális feladatot, de megint az a baj vele, hogy nem hagy elég teret az esetleges későbbi bővítés számára. Az újrahasznosíthatóság szempontjából az a jó, ha minél függetlenebbek egymástól az egyes modulok, azaz esetünkben a két panel. Ha függetlenek, akkor könnyen megoldható, hogy pl. ez a listás felület egy egészen más jellegű „csata-felület”-hez kapcsolódjon a későbbiekben.

Úgy lehet függetlenné tenni őket, hogy mindketten egy vezérlő osztályhoz csatlakoznak – ez a vezérlő könnyedén tudja majd cserélgetni az egyes panelokat.



Vagyis ahogy az ábrán is látható, a két panel független egymástól, és mindketten csak a vezérlővel vannak kapcsolatban.

Ez első hallásra talán bonyolultnak tűnik, de tapasztalhatja majd, hogy jóval kényelmesebbé is teszi a programozást, mert így minden lényeges funkció „egy kézben” összpontosul.

Még egy fontos kérdés: honnan ismeri majd egymást a vezérlő és az adott panel? A vezérlőnek mindkét panelt ismernie kell, vagyis csak ott lehet „bemutatni” őket egymásnak, ahol együtt szerepel mindkettő. Ez a hely a frame. Előbb azonban fel kell készíteni az osztályokat arra, hogy egyáltalán „megismerkedhessenek” egymással. Ez a következőt jelenti:
A vezérlő osztálynak majd két panelt kell irányítania, vagyis tudnia kell, melyik ez a két panel. Ezt lefordítva a programozás nyelvére, azt jelenti, hogy rendelkeznie kell két panel példánnyal, azaz két ilyen adattaggal:

```
private CsataPanel csataPanel;
private HarcosokPanel harcosokPanel;
```

Ezek vagy a konstruktorban vagy setter-rel kaphatnak értéket. Mindkét megoldás jó, a konstruktoros mellett az szól, hogy ekkor kevésbé lehet elfelejteni, hogy átadjuk a paneleket a vezérlőnek.

A paneleknek pedig azt kell tudniuk, hogy ki a vezér. Ezt ugyancsak lefordítva azt jelenti, hogy a paneleknek pedig rendelkeznie kell egy-egy `Vezerlo` típusú adattaggal:

```
private Vezerlo vezerlo;
```

Ez setter-rel kaphat értéket.

A frame-n tehát csak „össze kell ismertetni” őket:

```
private void vezerlesBeallitas() {
    Vezerlo vezerlo = new Vezerlo(csataPanel1, harcosokPanel1);
    harcosokPanel1.setVezerlo(vezerlo);
    csataPanel1.setVezerlo(vezerlo);
    vezerlo.start();
}
```

A `Vezerlo` osztály `start()` metódusába kerülnek majd az indításhoz szükséges teendők, ezt a vezérlésbeállító metódust pedig a frame `start()` metódusából hívjuk meg.

```
new HelloFrame().start();


private void start() {
    setVisible(true);
    vezerlesBeallitas();
}
```

Megjegyzés: Logikus lenne a vezérlő osztályt is singleton-ként kezelni. Próbálja meg így átalakítani.

Csatajelenetek:

Gombnyomás hatása:
A `HarcosokPanel` „csatába” gombja megnyomásának hatására a kiválasztott katonákat harcba küldjük. Ez a panel részéről nagyon egyszerű feladat, hiszen a harcosok kiválasztása után egyetlen dolga van: jelentse a vezérnek, hogy készen áll a csapat. A többi már a vezérlő dolga.

```
private void btnCsataActionPerformed(java.awt.event.ActionEvent evt) {
    List<Katona> valasztottKatonak = lstKatonak.getSelectedValuesList();
    vezerlo.csatabaAllit(valasztottKatonak);
    lstKatonak.clearSelection();
}
```

A `CsataPanel`-nek is van teendője a katonákkal: ki kell rajzolnia őket (természetesen a háttérkép kirajzoltatása után). Csakhogy ennél sokkal egyszerűbb a dolga (ténylegesen ő csak egy „rabszolga”), mégpedig összesen annyi, hogy kirajzolja azt, amit a vezér mond neki.
Vagyis: hívja meg a vezérlő osztály `rajzolas()` metódusát. Ahhoz, hogy a vezér elmondhassa a rajzolással kapcsolatos elképzeléseit, ismernie kell a rajzoló metódusokat. Ezek a `Graphics` osztály adott objektumán keresztül érhetőek el, vagyis a `rajzolas()` metódusban át kell adni a `paintComponent()` metódus `Graphics` típusú paraméterét.
A többi a vezérlő dolga.

FONTOS:
Így hívjuk meg a vezérlő rajzol metódusát (a panel `paintComponent()` metódusában):

```
if(vezerlo != null){
    vezerlo.rajzolas(g);
}
```

Evvel azt jelezzük, hogy csak akkor tudjuk meghívni a vezérlő osztály metódusát, ha egyáltalán létezik a vezérlő. Látszólag ez nem lényeges megjegyzés, hiszen úgy tűnik, enélkül is működik a program. Valóban, a probléma nem is itt keletkezik, hanem a frame design felületén. Ezt ugyanis már tervezési időben megpróbálja létrehozni, azaz már tervezési időben megpróbál rajzolni. Ekkor viszont még nincs vezérlő osztály, ezért hibaüzenetet kapunk. (Rossz esetben fel sem tudjuk húzni rá a panelt, vagy akár le is fagyhat. Egyébként érdemes akkor felhúzni a panelt, amikor még nem írtuk meg a `paintComponent()` metódust.)

Ennek a panelnek még egy feladata van: érzékelni, hogy ha rákattintottak, illetve továbbítani a kattintás helyét a vezérlőnek.

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
    vezerlo.kattintottak(evt.getX(), evt.getY());
}
```

Azt, hogy mit kezdjen evvel az információval, szintén a vezérlő dolga.

Láthatjuk, hogy a paneleknek meglehetősen kényelmes dolguk van, csak azt kell csinálniuk, amit a vezér mond nekik, illetve a velük történtekről értesíteniük kell a vezért. Tiszta rabszolga-sors, de a panelek nem bánják.

Essen még szó a `HarcosokPanel` további két feladatáról is: az egyik a már jól ismert metódus: átveszi a beolvasott modellt, és hozzárendeli a listafelülethez, a másikra majd a vezér kéri meg: módosítsa az eltalált harcos listabeli helyét. Persze, a panelnek fogalma sincs

róla, hogy ezt a harcost eltalálták, ő pusztán csak annyit tud, hogy egy harcost át kell sorolni a listában. Ez azt jelenti, hogy az illetőt ki kell törölni a listából (ténylegesen a modellből), és visszarakni a rendezésnek megfelelő helyére. Még pontosabban: akkor kell visszatenni, ha még él, egyébként nem.

```
public void listabaIr(RendezhetoListModel<Katona> katonaModell) {
    this.katonaModell = katonaModell;
    lstKatonak.setModel(katonaModell);
}

public void listaModositas(Katona katona) {
    katonaModell.removeElement(katona);
    if(katona.isElo()) katonaModell.addElement(katona,true);
}
```

Megjegyzés:
Mivel a `katonaModell` `RendezhetoListModel` típusú, ezért már a rendezés szerinti helyére is be lehet szúrni. Mivel ez egy saját készítésű modell, ezért nem biztos, hogy mindenre fel van készítve. Természetesen nekünk kell megírni az `addElement()` és a `removeElement()` metódust is. Nézze át mindkettőt a kiadott induló projekt modell osztályában.

Láthattuk tehát, hogy a panelek élete kicsit unalmas ugyan, de meglehetősen egyszerű. Annál több minden hárul a vezérlőre.

A vezérlő feladatai

Ténylegesen ő foglalkozik a harcosokkal. Ezek a feladatai:

- statikus adatok beállítása
- beolvasás kezdeményezése
- a kijelölt harcosok csatába állítása
- a harcosok kirajzolása
- a kattintás esemény tényleges kezelése

Ezek után nézzük meg a megvalósítás néhány további részletét.

Előtte még egy apró megjegyzés: A `sebzes` szó nem biztos, hogy a legjobb kifejezés. Több gyakorlaton is az `eletEro` elnevezést használtuk helyette, voltak, akik a `hp` kifejezést ajánlották inkább. Remélem, nem okoz problémát, hogy ennek ellenére meghagytam a saját megoldásban szereplő `sebzes` elnevezést, és most ezt másolom be ide.

A `Katona` osztály setterek/getterek nélkül:

Mivel rendezhető listamodellbe akarjuk rakni őket, ezért az osztálynak implementálnia kell a `Comparable` interfészt, és meg kell valósítania a `CompareTo()` metódust.

```
public class Katona implements Comparable<Katona>{
    private String nev;
    private static int utolsoIndex;
    private int sorSzam;
    private int sebzes;
    private boolean elo = true;

    private int kx, ky;
    private static int sugar;
    private Color szin;

    public Katona(String nev) {
        this.nev = nev;
        utolsoIndex++;
        sorSzam = utolsoIndex;
    }

    public void lott() {
        if (elo) {
            sebzes++;
        }
    }

    public void megLottek() {
        if (elo) {
            sebzes--;
            if (sebzes <= 0) {
                elo = false;
            }
        }
    }
    @Override
    public String toString() {
        return String.format("%d. %s (%d sebzés)", sorSzam, nev, sebzes);
    }

    public void rajzolas(Graphics g){
        g.setColor(szin);
        g.fillOval(kx-sugar, ky-sugar, 2*sugar, 2*sugar);
    }

    public boolean eltalaltak(int x, int y) {
        double tav = Math.sqrt((x-kx)*(x-kx) + (y-ky)*(y-ky));
        return (tav < sugar);
    }

    @Override
    public int compareTo(Katona t) {
        return t.sebzes - this.sebzes;
```

Az egyik utód-osztály (a többi is hasonló):

```
public class Lovas extends Katona{

    private static Color lovasSzin;

    public Lovas(String nev) {
        super(nev);
        this.setSzin(lovasSzin);
    }

    public static Color getLovasSzin() {
        return lovasSzin;
    }

    public static void setLovasSzin(Color lovasSzin) {
        Lovas.lovasSzin = lovasSzin;
    }
}
```

A gyártó osztály és gyártó metódus – egyelőre „fapadosan”, vagyis úgy, hogy a rangot stringként kezeljük:

```
public class KatonaGyar {

    private static KatonaGyar peldany;

    private KatonaGyar() {
    }

    public static KatonaGyar getInstance(){
        if(peldany == null){
            peldany = new KatonaGyar();
        }
        return peldany;
    }

    public Katona getKatona(String nev, String rang) {
        switch (rang) {
            case "lovas parancsnok":
                return new LovasParancsnok(nev);
            case "gyalogos parancsnok":
                return new GyalogosParancsnok(nev);
            case "lovas":
                return new Lovas(nev);
            case "gyalogos":
                return new Gyalogos(nev);
            case "zászlós" :
                return new Zaszlos(nev);
            default:
                return new Katona(nev);
        }
    }
}
```

Az adatbevitel:

```
public interface AdatInput {

    public RendezhetoListModel<Katona> katonaModellBevitel() throws Exception;
}

public class AdatBazisInput implements AdatInput {

    private Connection kapcsolat;

    public AdatBazisInput(Connection kapcsolat) {
        this.kapcsolat = kapcsolat;
    }

    @Override
    public RendezhetoListModel<Katona> katonaModellBevitel() throws Exception {
        RendezhetoListModel<Katona> katonaModell = new RendezhetoListModel<>();
        String sqlUtasitas = "SELECT * from KATONAK";

        String nev, rang;
        Katona katona;
        KatonaGyar rangGyar = KatonaGyar.getInstance();

        try (Statement utasitasObjektum = kapcsolat.createStatement();
                ResultSet eredmenyHalmaz
                = utasitasObjektum.executeQuery(sqlUtasitas);) {

            while (eredmenyHalmaz.next()) {
                nev = eredmenyHalmaz.getString("nev");
                rang = eredmenyHalmaz.getString("rang");
                katona = rangGyar.getKatona(nev, rang);
                katonaModell.addElement(katona, true);
            }
        }

        return katonaModell;
    }
}
```

A megoldás szebb lenne, ha string helyett enumot használnánk (ahogy a gyakorlaton is tettük). Most csak azokat a részleteket mutatom, ahol eltér az enumos megoldás a másiktól.

Adatbevitelkor:

```
while(eredmenyHalmaz.next()){
    nev = eredmenyHalmaz.getString("nev");
    rang = eredmenyHalmaz.getString("rang");
    //kivesszük a string-ből a fölösleges szóközöket
    rang = rang.replaceAll("\\s+","");
    KatonaGyar.RANGOK rangtipus =
                                    KatonaGyar.RANGOK.valueOf(rang);
    katona = rangGyar.getPeldany().getKatona(nev, rangtipus);
    katonaModell.addElement(katona, true);
}
```

A gyár módosított részletei:

```
public static enum RANGOK {gyalogos, gyalogosparancsnok, lovas,
                           lovasparancsnok, zaszlos}

public Katona getKatona(String nev, RANGOK rang){
    switch(rang){
        case gyalogos:{
            return new Gyalogos(nev);
        }
        case gyalogosparancsnok:{
            return new GyalogosParancsnok(nev);
        }
        case lovas:{
            return new Lovas(nev);
        }
        case lovasparancsnok:{
            return new LovasParancsnok(nev);
        }
        case zaszlos:{
            return new Zaszlos(nev);
        }
        default:{
            return new Katona(nev);
        }
    }
}
```

További megjegyzések:

1. A `default` ág nem a legszerencsésebb, ráadásul akkor egyáltalán nem használható, ha a `Katona` osztályt esetleg absztraktként kezeljük.
Szebb megoldás lenne ez:

```
default:
    throw new Exception();
```

Ez viszont kötelező kivételkezelést vonz maga után – ezt érdemes továbbdobni, hiszen a beolvasó metódus eleve továbbdob bármilyen hibát:

2. Most elsődleges cél a rendezhető listamodell használata volt, de az igazi MVC szemlélet szerint a katona példányokat egy List típusú listába kellene beolvasni, magát a modellt csak a panelen kellene használni. Ekkor ugyanis a beolvasó osztályt bármilyen más, de nem a mi modellünket alkalmazó projektben is lehetne használni.
Ekkor a `HarcosokPanel` `listabaIr()` metódusa így alakulna:

```
private RendezhetoListModel<Katona> katonaModell =
                                        new RendezhetoListModel<>();
public void listabaIr(List<Katona> katonaLista) {
    for (Katona katona : katonaLista) {
        katonaModell.addElement(katona, true);
    }
    lstKatonak.setModel(katonaModell);
}
```

A vezérlő osztály:

```
public class Vezerlo {

    private CsataPanel csataPanel;
    private HarcosokPanel harcosokPanel;

    private List<Katona> harcosok = new ArrayList<>();
    private int rajzMagassag;

    public Vezerlo(CsataPanel csataPanel, HarcosokPanel harcosokPanel) {
        this.csataPanel = csataPanel;
        this.harcosokPanel = harcosokPanel;
    }

    public void start() {
        try {
            beallitas();
            beolvasas();
        } catch (Exception ex) {
            Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void beallitas(){
        Katona.setSugar(Global.SUGAR);
        Gyalogos.setGyalogosSzin(Global.GYALOGOS_SZIN);
        GyalogosParancsnok.setGyalogosParancsnokSzin(Global.GYALOG_PARANCSNOK_SZIN);
        Lovas.setLovasSzin(Global.LOVAS_SZIN);
        LovasParancsnok.setLovasParancsnokSzin(Global.LOVAS_PARANCSNOK_SZIN);

        this.rajzMagassag = (int) (csataPanel.getHeight()*Global.ARANY);
    }

    private void beolvasas() throws Exception {
        AdatInput adatInput = AdatBazisInput.getPeldany();
        RendezhetoListModel<Katona> katonaModell = adatInput.katonaModellBevitel();
        harcosokPanel.listabaIr(katonaModell);
    }
```

```java
/**
 * Csatába küldi a kiválasztott katonákat, azaz mindegyiknek megmondja,
 * hogy hova álljon - ez a csatatér egy véletlen pontja.
 *
 * @param valasztottKatonak
 */
public void csatabaAllit(List<Katona> valasztottKatonak) {
    this.harcosok = valasztottKatonak;
    for (Katona katona : harcosok) {
        katona.setKx((int) (Math.random() * (csataPanel.getWidth()
                - 2 * Katona.getSugar()) + Katona.getSugar()));
        katona.setKy((int) (Math.random() * (csataPanel.getHeight()
                - rajzMagassag - Katona.getSugar()) + rajzMagassag));
    }
    // frissítjük a csatapanelt
    csataPanel.repaint();
}

/**
 * Megmondja, hogy ezeket a katonákat kell kirajzolni, azaz meghívja a
 * katonák rajzol() metódusát.
 *
 * @param g
 */
public void rajzolas(Graphics g) {
    for (Katona katona : harcosok) {
        katona.rajzolas(g);
    }
}

/**
 * Mi történjen, ha a panel jelzi, hogy rákattintottak az adott helyen.
 * Végignézi az összes harcost, és ha valakit eltaláltak, akkor a megadott
 * eséllyel vagy meglövik, vagy ő lőtt. Kérjük a harcosok panelt, hogy az
 * eredmény alapján módosítsa a listafelületet. Ha már nem él, akkor
 * kivesszük a harcosok közül.
 *
public void kattintottak(int x, int y) {
    for (Katona katona : harcosok) {
        if (katona.eltalaltak(x, y)) {
            if (Math.random() < SZAZALEK) {
                katona.megLottek();
            } else {
                katona.lott();
            }
            harcosokPanel.listaModositas(katona);
            if (!katona.isElo()) {
                harcosok.remove(katona);
            }

            // frissítjük a csatapanelt
            csataPanel.repaint();

            // Nagyon fontos ez a break, mert különben törlés után
            // összeomlik a ciklus.
            break;
        }
    }
}
}
```

```
    // Kevésbé szép, de biztonságos megoldás.
    //    public void kattintottak(int x, int y) {
    //        Katona torlendo = null;
    //        for (Katona katona : harcosok) {
    //            if(katona.eltalaltak(x,y)){
    //                if(Math.random() < SZAZALEK) katona.megLottek();
    //                else katona.lott();
    //                harcosokPanel.listaModositas(katona);
    //                if(!katona.isElo()){
    //                    torlendo = katona;
    //                }
    //                break;
    //            }
    //        }
    //        if(torlendo != null) harcosok.remove(torlendo);
    //        csataPanel.repaint();
    //    }
}
```

A kikommentezett ötlet akkor használható igazán (de ekkor a `break` nélkül), ha azt is elképzelhetőnek és megengedettnek tartjuk, hogy egy egérkattintással egyszerre több katona-pöttyöt is eltaláljunk. Ekkor egy listába kellene gyűjteni a törlendőket, majd a ciklus után törölni.