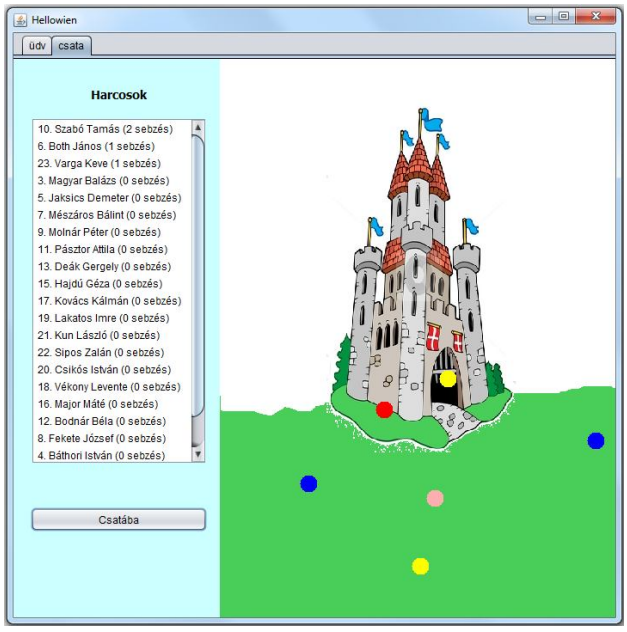
Hello Wien!

Egyre trendibb megünnepelni a „Hellowien”-t, nézzük meg, hogy mit is ünneplünk ekkor, és írjunk hozzá egy programot. 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.”



Az első tabulátor-felületen látható az üdvözlő oldal, közben pedig zene szól a háttérben. A második oldalon a csatát „szimuláljuk”. 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 240 pixel szélességű. Most csak egy-egy pötty jelzi a harcosokat, de ezt később lehet módosítani.

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 (minél erősebbek, annál nagyobb ez az érték). Mint az 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, a zászlós pink, 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 erősödik,

vagyis a sebzés értéke eggyel nő, ha meglőtték, akkor eggyel csökken, és ha negatívvá válik, akkor szegény katona meghal. Csak élő katona lőtt, és őt lehet meglőni.

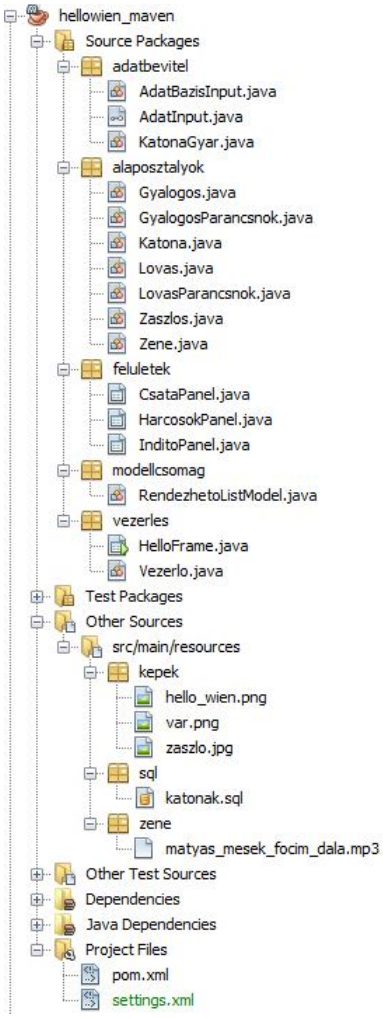
A „Csatába” feliratú gomb hatására a listából kijelölt katonák harcba indulnak. Ez azt jelenti, hogy a csatatéren véletlen helyeken megjelennek az őket jellemző pöttyök. (Mivel akkor még nem volt légierő, ezért lehetőleg ne a levegőben jelenjenek meg, hanem – a háttérképtől függő magasságig, mondjuk, a panel 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 vagy meglövik, vagy ő lő. (Ezt a véletlen dönti el.)

Természetesen ennek megfelelően állandóan változik majd a listában látható sorrend, illetve ha egy katona meghal, akkor a listából is, csatatérről is töröljük.

Egy lehetséges megoldás:



A Maven projektek forrás állományai egy előre definiált mappa struktúrában vannak. Egy Java Maven projekt struktúrája alapvetően a következőképpen néz ki:

(A resources mappákat legegyszerűbben valamilyen fájlkezelővel hozhatjuk létre.) A projekt szerkezete kicsit eltérően jelenik meg NetBeans vagy Eclipse környezetben, a részletes ábrán egy NetBeans környezet látható.

Esetünkben a pom.xml fájl:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.ptemik</groupId>
    <artifactId>hellowien_maven</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Függőségek: a zenéért felelős javazoom és a derby -->

        <!-- https://mvnrepository.com/artifact/javazoom/jlayer -->
        <dependency>
            <groupId>javazoom</groupId>
            <artifactId>jlayer</artifactId>
            <version>1.0.1</version>
        </dependency>

        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derby</artifactId>
            <version>10.14.1.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derbyclient</artifactId>
            <version>10.14.1.0</version>
        </dependency>

    </dependencies>
    <!-- Futtatható .jar állomány létrehozásának beállítása -->
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <archive>
                        <manifest>
                            <mainClass>vezerles.HelloFrame</mainClass>
                        </manifest>
```

```
                </archive>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

</project>
```

Innen letölthető: pom.xml – de a megadott adatok között is szerepel. (A megoldás írásakor ezek voltak az aktuális verziók, de esetleg érdemes utánanézni, hogy azóta nincsenek-e újabbak.)

A projekt megbeszélésében és megoldásában haladjunk a következő vázlatpontok alapján:

1. Létrehozzuk a felületeket. Az első oldal kialakításában semmi újdonság nincs, a második oldalt viszont 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 tovább-fejlesztéskor lehet érdekes. A lényegesebb szempont az, hogy ennek kapcsán arról is szó eshet, hogyan lehet kapcsolatot teremteni két panel között. (A két panel: a harcosok listáját tartalmazó a `HarcosokPanel`, a csatatér a `CsataPanel`-en van.)

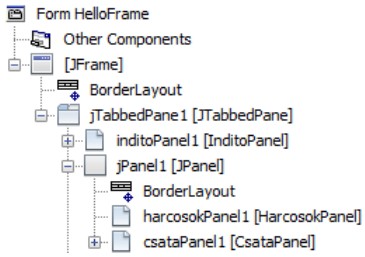
2. Megoldjuk az adatbevitelt. Ehhez több dolgot kell végiggondolnunk:

- Hogy épüljön fel a Katona osztály?
- Hogyan oldjuk meg a rangok kezelését?
- Hogyan olvassunk be a megadott adatbázisból?

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

4. Zenét rendelünk az üdvözlő felülethez. Ezt a lehető legegyszerűbben oldjuk meg, hiszen a zene elsősorban azért került be a feladatba, hogy indokoltabb legyen a Maven használata.

Fejtsük ki egy kicsit bővebben az előző vázlatpontokat!



1. A **felületek** kialakításához egyetlen megjegyzés: a második tabulált felületre előbb egy sima panel kerül, ezt állítsuk border layout-ra, majd erre illesszük rá (óvatosan) a két panelt.

2. 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, illetve 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 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.

Lássuk az eddigiekhez szükséges kódot!

A `Katona` osztály – mivel rendezhető listamodellbe kerülnek a példányok, ezért összehasonlíthatóknak kell lenniük (+ setterek, getterek):

```
public class Katona implements Comparable<Katona>{

    private String nev;
    private static int utolsoSorszam;
    private int sorszam;

    private boolean elo = true;
    private int sebzes;

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

    public Katona(String nev) {
        this.nev = nev;
        utolsoSorszam++;
        sorszam = utolsoSorszam;
    }

    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);
    }

    protected void setSzin(Color szin) {
        this.szin = szin;
    }

    @Override
    public int compareTo(Katona o) {
        return o.getSebzes() - this.getSebzes();
    }
```

Az egyik utódosztály (a többi is hasonló):

```
public class Gyalogos extends Katona{

    private static Color gyalogosSzin;

    public Gyalogos(String nev) {
        super(nev);
        szinBeallitas();
    }

    public static Color getGyalogosSzin() {
        return gyalogosSzin;
    }

    public static void setGyalogosSzin(Color gyalogosSzin) {
        Gyalogos.gyalogosSzin = gyalogosSzin;
    }

    private void szinBeallitas() {
        this.setSzin(gyalogosSzin);
    }
}
```

Egy apró **megjegyzés**: A `sebzes` szó nem biztos, hogy a legjobb kifejezés. Esetleg használhatnánk helyette az `eletEro` elnevezést, vagy a számítógépes játékokban használt `HP` kifejezést, de a mostani megoldás bemutatásakor maradunk a `sebzes` elnevezésnél, de persze, saját megoldásban nyugodtan használhat mást.

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 utódtípus 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 gyártó osztály és gyártó metódus – egyelőre „fapadosan”, vagyis úgy, hogy a rangot `String`-ként 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 List<Katona> katonaListaBevitel() throws Exception;
}
```

```
public class AdatBazisInput implements AdatInput {

    private Connection kapcsolat;

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

    @Override
    public List<Katona> katonaListaBevitel() throws Exception {
        List<Katona> katonak = new ArrayList<>();
        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);
                katonak.add(katona);
        }
    }
    return katonak;
}
```

A megoldás szebb lenne, ha `String` helyett `enum` típust használnánk. Most csak azokat a részleteket mutatjuk, 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);
        }
    }
}
```

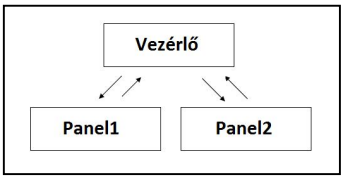
Megjegyzés: A feladat szerint most csak olvasni kell az adatbázisból, ezért nem használtuk a DAO tervezési mintán alapuló interfészt, de egy esetleges továbbfejlesztéshez már kellene.

Adatbevitel és megjelenítés:

Mint megbeszéltük, most azt a változatot választjuk, amely két külön panelen kezeli a lista-felületet 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 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, ahogy már a korábbi megoldásokban is tettük, vagyis úgy, 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.

A vezérlőnek mindkét panelt ismernie kell, vagyis csak ott lehet „bemutatni” őket egymásnak, ahol együtt szerepel mindkettő. Ez a hely továbbra is 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 (amit pedig a `main()` metódusból).

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

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

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

A `Vezerlo` osztály ide tartozó része:

```
public class Vezerlo {

    private final String SQL_ELERES ="/sql/katonak.sql";
    private final String CHAR_SET = "UTF-8";

    private final int SUGAR = 10;
    private final Color GYALOGOS_SZIN = Color.yellow;
    private final Color GYALOG_PARANCSNOK_SZIN = Color.green;
    private final Color LOVAS_SZIN = Color.red;
    private final Color LOVAS_PARANCSNOK_SZIN = Color.blue;
    private final Color ZASZLOS_SZIN = Color.pink;

    // a panel magasságának ilyen arányáig engedjük fel a katona-pöttyöket.
    private final double ARANY = 0.5;

    // max. ennyi százalék annak esélye, hogy meglövik a katonát
    private final double SZAZALEK = 0.4;

    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(SUGAR);
        Gyalogos.setGyalogosSzin(GYALOGOS_SZIN);
        GyalogosParancsnok.setGyalogosParancsnokSzin(GYALOG_PARANCSNOK_SZIN);
        Lovas.setLovasSzin(LOVAS_SZIN);
        LovasParancsnok.setLovasParancsnokSzin(LOVAS_PARANCSNOK_SZIN);
        Zaszlos.setZaszlosSzin(ZASZLOS_SZIN);
        this.rajzMagassag = (int) (csataPanel.getHeight() * ARANY);
    }
```

Az adatbázis természetesen külön adatbázis-szerveren is lehetne, előre elkészítve, de most úgy oldjuk meg, hogy csak a létrehozásához szükséges SQL fájl van megadva, az adatbázis pedig a program első futtatásakor jön létre.

```
private void beolvasas() throws Exception {
    Connection kapcsolat = kapcsolodas();
    AdatInput adatInput = new AdatBazisInput(kapcsolat);
    List<Katona> katonak =
            adatInput.katonaListaBevitel();
    harcosokPanel.listabaIr(katonak);
}

private Connection kapcsolodas() throws ClassNotFoundException, SQLException {
    Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
    String url = "jdbc:derby:HelloDB;create=true;";

    Connection kapcsolat = DriverManager.getConnection(url);

    String sqlVaneMarTabla =
            "select * from SYS.SYSTABLES where tablename = 'KATONAK'";

    try (Statement utasitasObj = kapcsolat.createStatement();
            ResultSet rs = utasitasObj.executeQuery(sqlVaneMarTabla)) {

        if (!rs.next()) {
            adatBazisLetrehozas(utasitasObj);
        }
    }
    return kapcsolat;
}

private void adatBazisLetrehozas(Statement utasitasObj) throws SQLException {
    try(InputStream ins = this.getClass().getResourceAsStream(SQL_ELERES);
            Scanner sc = new Scanner(ins, CHAR_SET)){
        String sor;
        while(sc.hasNextLine()){
            sor = sc.nextLine();
            System.out.println(sor);
            utasitasObj.execute(sor);
        }
    } catch (IOException ex) {
        Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

A `HarcosokPanel` osztály hivatkozott metódusa:

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

Beolvasáskor még nincsenek sebzések, vagyis nem fontos rendezve beszúrni, ez is elég lenne: `katonaModell.AddElement(katona)`.

3. 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 btnCsatabaActionPerformed(java.awt.event.ActionEvent evt) {
        List<Katona> valasztottak = lstKatonak.getSelectedValuesList();
        vezerlo.harcbaKuld(valasztottak);
        lstKatonak.clearSelection();
    }
```

A vezérlő hivatkozott metódusának bemutatása előtt még néhány mondat:

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.

A `CsataPanel` ide vonatkozó része:

```
public class CsataPanel extends javax.swing.JPanel {

    private final String VARKEP_ELERES = "/kepek/var.png";
    private Image varKep =
            new ImageIcon(this.getClass().getResource(VARKEP_ELERES)).getImage();
    private Vezerlo vezerlo;

    public CsataPanel() {
        initComponents();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int kezdoX =0, kezdoY = 0,
            szelesseg = this.getWidth(),
            magassag = this.getHeight();
        g.drawImage(varKep, kezdoX, kezdoY, szelesseg, magassag, null);

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

FONTOS: Azért így, azaz null-vizsgálattal hívjuk meg a vezérlő rajzol metódusát, mert 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 a fejlesztő környezet, 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 le is fagyhat. Érdemes akkor felhúzni a panelt, amikor még nem írtuk meg a `paintComponent()` metódust.)

Megjegyzés: Az `InditoPanel` háttérrajzolása ugyanolyan, mint az itteni, csak értelemszerűen más képet kell kirajzolnia.

A `Vezerlo` osztály hivatkozott metódusai:

```
/**
 * Véletlenszerűen beállítja a harcba küldött katonák pozícióját, majd
 * frissíti a rajzot.
 *
 * @param valasztottak
 */
public void harcbaKuld(List<Katona> valasztottak) {
    harcosok = valasztottak;
    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));
    }
    frissit();
}

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

private void frissit() {
    csataPanel.repaint();
}
```

„Katonára” kattintás:

A csata panelnek még egy feladata van: érzékelni, 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, már 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. A panelek élete kicsit unalmas ugyan, de meglehetősen egyszerű. Annál több minden hárul a vezérlőre.

A `Vezerlo` osztályban a kattintás hatására azt kell vizsgálni, hogy eltaláltuk-e valamelyik harcoló katonát (vagyis az őt jelző pöttyre kattintottunk-e). Ha igen, akkor véletlenszerűen vagy meglőtték, vagy ő lőtt, és ha már nem él, akkor ki kell törölni a harcosok közül. A törlés veszélyes művelet, ezért ha törlünk valakit a ciklus hatóköréből, akkor azonnal ki is kell lépnünk a ciklusból. (A másik, de nehézkesebb, lehetőség: megjegyezzük a törlendő példányt vagy annak indexét, és a ciklus lefutása után töröljük.) A `Vezerlo kattintottak()` metódusa:

```
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.listaFrissites(katona);

            if (!katona.isElo()) {
                harcosok.remove(katona);
                frissit();
            }
            break;
        }
    }
}
```

A `Katona` osztály `eltalaltak()` metódusa azt vizsgálja, hogy a kattintás helye a (kx, ky) középpontú körön belül van-e:

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

A `HarcosokPanel` hivatkozott metódusa:

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

Megjegyzés: Mivel a `katonaModell RendezhetoListModel` típusú, ezért azonnal be lehet szúrni az adott elemet a rendezés szerinti helyére. Mivel ez egy saját készítésű modell, ezért nekünk kell megírni az `addElement()` és a `removeElement()` metódust is. Ezeket önállóan is megírhatja, de le is töltheti a feladatkiírásnál megadott helyről. Ez esetben nézze át és értse meg a letöltött modell osztályt.

4. Zene

Mint már megjegyeztük, a zene nem kap központi szerepet a feladatmegoldásban, de persze, ha kedve van hozzá, nézzen utána alaposabban. Most épp csak annyiról lesz szó, hogy hogyan lehet lejátszani egy mp3 fájlt.

A zenelejátszáshoz egy külön szálat indítunk (szálkezelésről később lesz szó, most csak előlegezzük meg a legalapvetőbb ismereteket).

Egy szálat a `Thread` osztályból származtatunk (ennek többféle módja is lehet, most örökítjük onnan), és az osztály `run()` metódusában írjuk le, hogy mi is a szál teendője.

Bár a feladat szerint csak végig akarjuk játszani a zenét, de a most bemutatandó `Zene` osztályban a leállítás lehetőségét is megadjuk. Az osztály kódja (getterek nélkül):

```
public class Zene extends Thread {

    private String zenefajlEleres;
    private InputStream stream;
    private Player player;
    private boolean szolhat = true;

    public void setZeneFajlEleres(String zenefajlEleres) {
        try {
            this.zenefajlEleres = zenefajlEleres;
            stream = this.getClass().getResourceAsStream(zenefajlEleres);
            player = new Player(stream);
        } catch (JavaLayerException ex) {
            Logger.getLogger(Zene.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

```
@Override
public void run() {
    try {
        player.play();
    } catch (JavaLayerException ex) {
        Logger.getLogger(Zene.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void leall() {
    player.close();
}
```

A zenével kapcsolatos osztályok nincsenek a standard Java csomagok között, külön kell importálni őket:

```
import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;
```

Ehhez szükség van a javazoom csomagra – ezt töltötte le a Maven a pom.xml fájlban megadott információk alapján (és persze, a Derby kezeléséhez szükséges csomagokat is).

A zene a program indulásakor azonnal megszólal, vagyis a vezérlő osztály `start()` metódusából indítjuk.

```
private final String ZENEFAJL_ELERES = "/zene/matyas_mesek_focim_dala.mp3";
private Zene zene;

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

public void zeneBekapcsolas() {
    if(!zene.isAlive()){
        zene.setZeneFajlEleres(ZENEFAJL_ELERES);
        zene.start();
    }
}
```

Ha valóban csak annyit szeretnénk, hogy közbeavatkozás nélkül lejátsszuk a megadott zenét, akkor nincs szükség a `zene` nevű mezőre, elég, ha csak lokálisan deklaráljuk, sőt, maga a `zeneBekapcsolas()` metódus is egyszerűsödhet:

```
public void zeneBekapcsolas() {
    Zene zene = new Zene();
    zene.setZeneFajlEleres(ZENEFAJL_ELERES);
    zene.start();
}
```

Az eredeti verzióra akkor van szükség, ha esetleg közben le is akarjuk állítani a zenét, vagy netalántán újraindítani. Ekkor figyelni kell, hogy él-e már a szál (`isAlive()`), mert egy szálat csak egyszer lehet elindítani.

Még egy érdekes „kísérlet”: próbálja ki (de ne lepődjön meg a hatáson), hogy a zene példánynak nem a `start()` metódusát hívja meg (egy új szálat mindig így hívunk meg), hanem közvetlenül a `run()` metódusát. Ekkor nem indul új szál, hanem az eredeti program-szál része lesz a zene, és addig semmit nem tudunk kezdeni a megnyitott, fehér felületű projekttel, amíg véget nem ér a zene. Csak ez után jelennek meg a képek, ez után tudunk bármit is csinálni – még bezárni sem tudjuk az alkalmazást (csak feladatkezelőből).

Utolsó **megjegyzés**: A pom.xml fájlban azt is beállítottuk, hogy a build során futtatható .jar állomány jöjjön létre. Ezt az állományt a projekt *target* mappájában találja meg, *hellowien_maven-1.0-SNAPSHOT-jar-with-dependencies.jar* néven. (Előfordulhat, hogy NetBeans hiba miatt csak a jar futtatásakor hallatszik a zene.)