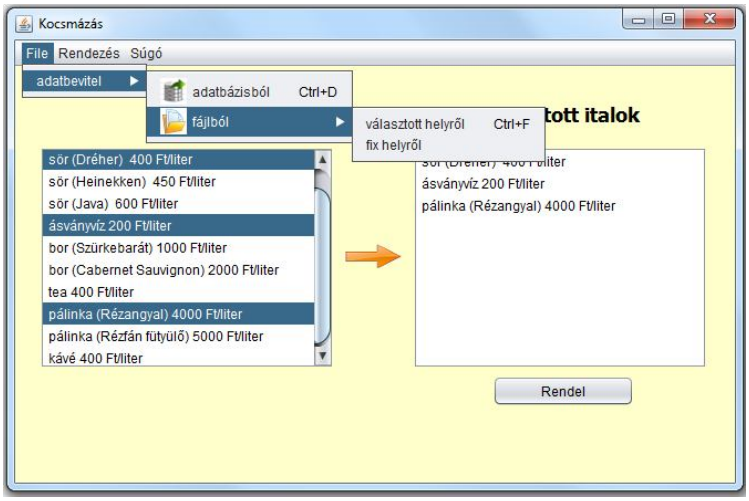
5. gyakorlat: a 4. gyakorlat folytatása: olvasás, rendezés, tartalmazás

Feladat:

Kocsmázás ürügyén sok mindent szerettem volna megbeszélni, egyelőre csak egy részét sikerült, most csak ezt beszéljük meg.



A 650*400-as belső felületű alkalmazásban választhassuk ki, hogy honnan olvassuk be az adatokat.

Ahogy látható, az itallapon szereplő italok között vannak alkoholos és nem alkoholos italok is.
Az ital definiálásakor meg kell adnunk a fajtáját (bor, tea, víz, stb), vonalkódját és literenkénti árát.

Az alkoholos italt a fentieken kívül még egy márkanév és az alkoholfok is jellemzi.

Most megbeszéljük a választott helyről való olvasást és a rendezéseket.

Olvasás:

Ahhoz, hogy ki tudjuk választani a fájlt, szükségünk van egy `JFileChooser` típusú példányra. Ezt elvileg a palettáról is áthúzhatnánk (de vigyázzon rá, hogy vagy a design felület alatti fehér területen ejtse el, vagy a navigációs panel Other Components elemén), de szerintem jóval kényelmesebb, ha kódból adjuk meg. Hogy mezőként vagy lokálisan, az nyilván attól függ, hogy hány metódusban akarjuk használni.

A fájlválasztó megnyitásakor azt kell figyelni, hogy jóváhagytuk-e a választást (approve), és ha igen, akkor kiválaszthatjuk a fájlt. Természetesen itt is kell a kivételkezelés, hiszen elképzelhető, hogy nem is létezik az adott fájl, vagy hibásat választottunk, de most eleve előírtuk az általános kivételkezelést, emiatt evvel most nem kell külön törődnünk.

Mivel itt már meghatároztuk a kiválasztott fájlt, célszerű lenne közvetlenül innen olvasni (vagyis az inputstream kihagyásával). Ennek megfelelően át is írtuk az előző órán megírt fájlból olvasó osztályt (gyakorlatilag „visszaírtuk” a korábban ismert formára). Ez viszont elrontja a fix helyről való olvasás lehetőségét. Az elérési utat azonban nem csak az inputstream segítségével tudjuk megadni, hanem más módon is. (Hogy adott helyről való olvasás esetén a két megadási mód közül melyik az egyszerűbb, nyilván szubjektív, ki-ki döntse el magának.)

Ezeket figyelembe véve a beolvasó osztály, illetve alkalmazása a két különböző esetre:

```java
public class FajlbolInput implements AdatInput {

    private File adatFajl;
    private final String CHAR_SET = "UTF-8";

    public FajlbolInput(File adatFajl) {
        this.adatFajl = adatFajl;
    }

    @Override
    public List<Ital> italLista() throws Exception {
        // lehet úgy is, hogy üresen - vagy a figyelmeztető üzenettel hagyjuk
        // valamelyik metódust
        List<Ital> italok = new ArrayList<>();
        // mivel az inputstream is és a scanner is lezárandó objektum, ezért
        // érdemes a try fejében megnyitni, ekkor automatikus a lezárás

        // Ezt a két sort cseréltük ki közvetlen fájlból való olvasásra
//      try (InputStream ins = this.getClass().getResourceAsStream(italEleres);
//              Scanner fajlScanner = new Scanner(ins, CHAR_SET)){
        try (Scanner fajlScanner = new Scanner(adatFajl, CHAR_SET)) {
            String sor;
            String[] adatok;
            Ital ital = null;
            while (fajlScanner.hasNextLine()) {
                sor = fajlScanner.nextLine();
                adatok = sor.split(";");
                if (!sor.isEmpty()) {
                    adatok = sor.split(";");
                    if (adatok.length == 4) {
                        ital = new Ital(adatok[0], adatok[1],
                                Integer.parseInt(adatok[2]),
                                Double.parseDouble(adatok[3]));
                    }
                    if (adatok.length == 6) {
                        ital = new AlkoholosItal(adatok[0], adatok[1],
                                Integer.parseInt(adatok[2]),
                                adatok[3], Double.parseDouble(adatok[4]),
                                Double.parseDouble(adatok[5]));
                    }
                }
                italok.add(ital);
            }
        }
        return italok;
    }
```

A `KocsmaPanel` osztályban:

```
    private final String ITAL_ELERES = "/adatok/arlista.txt";
    private DefaultListModel<Ital> italModel;

    public boolean valasztottHelyrol() {
        JFileChooser fajlValaszto = new JFileChooser(new File("."));
        if (fajlValaszto.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            try {
                AdatInput adatInput
                        = new FajlbolInput(fajlValaszto.getSelectedFile());
                adatBevitel(adatInput);
                return true;
            } catch (Exception ex) {
                Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                        null, ex);
            }
        }
        return false;
    }

    private void adatBevitel(AdatInput adatInput) throws Exception {
        italModel = adatInput.italModell();
        lstItallap.setModel(italModel);
    }

    public boolean fixFajlbol() {

        try {
            File adatFajl
                 = new File(this.getClass().getResource(ITAL_ELERES).toURI());

            AdatInput adatInput = new FajlbolInput(adatFajl);
            // Ezt cseréltük ki a korábbi változatból:
//            AdatInput adatInput = new FajlbolInput(ITAL_ELERES);
            adatBevitel(adatInput);
            return true;
        } catch (Exception ex) {
            Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                    null, ex);
            return false;
        }
    }
```

Rendezés:

Egyelőre még csak a default listamodellt vettük, amelyet nem tudunk beépített metódussal rendezni. Írhatnánk rá saját rendezést, de kényelmesebb, ha ideiglenesen átrakjuk a modell elemeit egy listába, erre alkalmazzuk a sort() metódust, majd visszarakjuk az elemeket a modell-be.
Ahhoz, hogy alkalmazni tudjuk a `Collections.sort()` metódust, vagy az `Ital` osztályt kellene `Comparable` típusúvá tennünk, vagy kellene egy `Comparator` típusú rendező osztály. Most ez utóbbit választjuk, de ha végzett evvel a megoldással, akkor próbálja meg úgy is, hogy az Ital osztályt alakítja át – nagyon hasonlóan lehet, mint most ezt.

Bármi legyen is a rendezési szempont, ha az meg van fogalmazva a `Rendezes` osztályban, akkor a modell elemeinek rendezése mindig így néz ki:

```
private void rendez() {
    List<Ital> italok = Collections.list(italModel.elements());
    Collections.sort(italok,new Rendezes());
    italModel.clear();
    for (Ital ital : italok) {
        italModel.addElement(ital);
    }
}
```

A fő kérdés most az, hogy hogyan írjuk meg ezt a `Rendezes` osztályt.

Mivel háromféle szempont szerint szeretnénk rendezni, ezért a szempontokat célszerű `enum`-ként deklarálni, és majd a választott szempontot ezek közül határozhatjuk meg.
Azt is tudnunk kell, hogy növekvően vagy csökkenően szeretnénk rendezni. Mivel ez két lehetőség, érdemes `boolean`-ként kezelni. Megállapodhatunk abban, hogy pl. a `true` érték jelenti azt, hogy növekvően rendezzük. De ezt kicsit kényelmetlen fejben tartani, ezért szokásos megoldás, hogy inkább elnevezzük valami érthető szóval. Esetünkben `NOVEKVOEN`-nek neveztük. Erre ebben a feladatban egyáltalán nincs szükség, de így felkészítettük az osztályt arra, hogy később máshonnan is könnyen meg lehessen hívni.

Mivel `final` értékek, ezért publikusnak is deklarálhatóak, de figyelem **CSAK** akkor lehet publikus egy mező, ha `final`, vagyis nem változtatható.

Még egy apróság, mielőtt rátérnénk a kódrészletre. Mivel majd meg kell adni a rendezési szempontot is, és a rendezés módját is (növekvő vagy csökkenő), célszerű a két setter helyett egyet írni úgy, hogy abban mindkét paraméter értékét meg kelljen adni.

A `compare()` metódusban a különböző szempontok szerinti összehasonlítást adjuk meg.

Az osztály getterek nélküli része további magyarázat nélkül:

```java
public class Rendezes implements Comparator<Ital>{

    public enum Szempont{FAJTA, AR, ALKOHOLFOK}

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

    private static Szempont valasztottSzempont;
    private static boolean miModon;

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

    @Override
    public int compare(Ital o1, Ital o2) {
        switch (valasztottSzempont) {
            case FAJTA:
                return miModon ? o1.getFajta().compareTo(o2.getFajta())
                        : o2.getFajta().compareTo(o1.getFajta());
            case AR:
                return miModon ? o1.getLiterAr() - o2.getLiterAr()
                        : o2.getLiterAr() - o1.getLiterAr();
            case ALKOHOLFOK: {
                double o1fok, o2fok;
                o1fok = (o1 instanceof AlkoholosItal)
                        ? ((AlkoholosItal) o1).getAlkoholFok() : 0;
                o2fok = (o2 instanceof AlkoholosItal)
                        ? ((AlkoholosItal) o2).getAlkoholFok() : 0;
                return (int) (miModon ? Math.signum(o1fok - o2fok)
                        : Math.signum(o2fok - o1fok));
            }
            default:
                return 0;
        }
    }
```

A rendezés meghívása pl. ár szerinti rendezés esetén:

`InditoFrame`:

```java
    private void rdbNevsorSzerintMenuPontActionPerformed(java.awt.event.ActionEvent evt) {
        kocsmaPanel1.nevsorSzerint(chkNovekvolegMenuPont.isSelected());
    }
```

`KocsmaPanel`:

```java
    public void arSzerint(boolean novekvo) {
        Rendezes.setValasztottSzempont(Rendezes.Szempont.AR, novekvo);
        rendez();
    }
```

Azonban volt még egy feladat: ha alkoholfok szerint rendezünk, akkor a listafelületen jelenjen meg az alkoholos ital foka is, egyébként ne.

Ezt úgy lehet megoldani, hogy az `AlkoholosItal` osztály `toString()` metódusát felkészítjük az alternatívára, vagyis arra, hogy egy logikai változó értékétől függően vagy ilyen, vagy olyan alakú stringet adjon vissza. Mivel az összes alkoholos italt vagy alkoholfokkal vagy anélkül kell kiíratni, ezért ez a változó statikus.

Az `AlkoholosItal` osztály ide vonatkozó kódrészlete:

```
    private static boolean rendezve;

    @Override
    public String toString() {
        String temp = String.format("%s (%s) %4d Ft/liter", this.getFajta(),
                                                this.markaNev, this.getLiterAr());
        if (rendezve) {
            return temp + " (" + alkoholFok + "°)";
        }
        return temp;
    }
```

Ezt a módosítást figyelembe véve a metódusok így alakulnak:

```
    public void nevsorSzerint(boolean novekvo) {
        AlkoholosItal.setRendezve(false);
        Rendezes.setValasztottSzempont(Rendezes.Szempont.FAJTA, novekvo);
        rendez();
    }

    public void alkoholfokSzerint(boolean novekvo) {
        AlkoholosItal.setRendezve(true);
        Rendezes.setValasztottSzempont(Rendezes.Szempont.ALKOHOLFOK, novekvo);
        rendez();
    }
```

Természetesen az árra vonatkozó metódusban is hamisra kell állítani az alkoholos ital rendezettségét.

A rendezéssel kapcsolatban még egy teendőnk van: a növekvő/csökkenő módon való rendezésre vonatkozó menüpont kezelése. Ez nagyon egyszerű, csak azt kell figyelni, hogy melyik rádiógomb van bekapcsolva, és a hozzá tartozó metódust kell meghívnunk:

```
    private void chkNovekvolegMenuPontActionPerformed(java.awt.event.ActionEvent evt) {
        if (rdbAlkoholfokSzerintMenuPont.isSelected()) {
            kocsmaPanel1.alkoholfokSzerint(chkNovekvolegMenuPont.isSelected());
        }
        if (rdbArSzerintMenuPont.isSelected()) {
            kocsmaPanel1.arSzerint(chkNovekvolegMenuPont.isSelected());
        }
        if (rdbNevsorSzerintMenuPont.isSelected()) {
            kocsmaPanel1.nevsorSzerint(chkNovekvolegMenuPont.isSelected());
        }
    }
```

Tartalmazás:

Ennek a feladatrésznek az elsődleges célja a tartalmazás megbeszélése. A feladat szerint a nyíllal jelölt gomb (`btnValasztas`) hatására „válasszunk” az itallapról, és a választott italok jelenjenek meg a választott italok között (jobboldali listafelület).

A feladatot úgy is lehet értelmezni, hogy a baloldali listán lévő szövegek közül a kiválasztottak kerüljenek át a jobboldaliba. Látszólag ennyit kér a feladat. Ez roppant egyszerű, a kiválasztott italokat át kell rakni a jobboldali listafelülethez tartozó modellbe (`rendeltItalModel`).

A feladat azonban másképpen is értelmezhető, sőt, a feladatsor második feladatát eleve így lehet majd megoldani, vagyis úgy, hogy nem az eredeti példányokat adjuk át, hanem azoknak a másolatát, vagyis a kiválasztott ital példányok paraméterei alapján létrehozunk egy másik ital példányt. Ez akár „életszagú” is lehet, hiszen a kocsmában sem azt az italt kérjük, ami pl. a szomszéd asztalán van, hanem egy olyat. Vagy ha úgy képzeljük el az „itallapot”, hogy nem csak a szöveget látjuk, hanem a pulton mindegyikhez ki van rakva egy-egy pohárnyi minta, akkor szintén nyilvánvaló, hogy nem azt kérjük, ami a pulton van, hanem csak egy olyat.
De tény, hogy pillanatnyilag főleg azért csináltuk így, hogy a tartalmazást is meg tudjuk beszélni, de nyilván az is motivált, hogy ez jó előkészítése a feladatsor következő feladatának.

Most annyit szeretnénk, hogy a választottak közül minden féle ital csak egyszer jelenjen meg (ezt majd a második feladat oldja fel, ahol majd azt is ki kell íratni, hogy melyikből hány pohárral kértünk).

A `KocsmaPanel` ide vonatkozó kódrészlete:

```
private void btnValasztasActionPerformed(java.awt.event.ActionEvent evt) {
    List<Ital> valasztottak = lstItallap.getSelectedValuesList();

    Ital rendeltItal;
    for (Ital ital : valasztottak) {
            rendeltItal = (ital instanceof AlkoholosItal)
                ? new AlkoholosItal(ital.getFajta(), ital.getVonalKod(),
                                    ital.getLiterAr(),
                                    ((AlkoholosItal) ital).getMarkaNev(),
                                    ((AlkoholosItal) ital).getAlkoholFok())
                : new Ital(ital.getFajta(), ital.getVonalKod(),
                           ital.getLiterAr());

            if (!(rendeltItalModel.contains(rendeltItal))) {
            rendeltItalModel.addElement(rendeltItal);
        }
    }
}
```

Ha azonban csak ennyit csinálunk, akkor a `contains()` metódus „nem működik”. Pontosabban nagyon is működik, hiszen mivel mindig új példányt hozunk létre, nyilván nem lesz az új példány az eddigiek között. Ha azt szeretnénk, hogy az ugyanolyanok ne kerüljenek be duplán, akkor meg kell mondanunk, hogy mit jelent az, hogy „ugyanolyan”. Ehhez összesen annyi kell, hogy az `Ital` osztályban generáljuk az `equals()` és `hashCode()` metódusokat. A generálás

során be kell állítani, hogy mikor tekintünk egyformának két elemet. Órán a fajta és a vonalkód egyezőségét írtuk elő.

Megjegyzés:
A másolatot egyszerűbben is létre lehet hozni (csak akkor nem gyakoroltuk volna a típuskényszerítést ☺).

Különösebb magyarázat nélkül mutatom meg a megoldást.

Az `Ital` osztályba még beszúrjuk ezt a kódrészletet:

```
private Ital masolat;
private Ital (Ital ital){
    masolat = ital;
}

public Ital getMasolat() {
    Ital temp = new Ital(this);
    return temp.masolat;
}
```

vagyis egy olyan konstruktort, amelynek `Ital` típusú paramétere van. Ez azért `private`, hogy csak az osztályon belülről lehessen meghívni – vagyis esetünkben a másolatot visszaadó metódusban.

A `KocsmaPanel` módosított metódusa:

```
private void btnValasztasActionPerformed(java.awt.event.ActionEvent evt) {
    List<Ital> valasztottak = lstItallap.getSelectedValuesList();

    Ital rendeltItal;
    for (Ital ital : valasztottak) {
            rendeltItal = ital.getMasolat();
            if (!(rendeltItalModel.contains(rendeltItal))) {
            rendeltItalModel.addElement(rendeltItal);
        }
    }
}
```