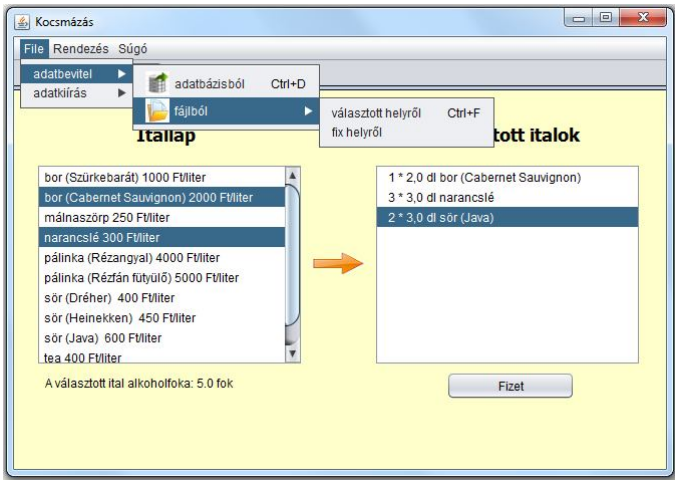
Kocsmázás

Az eddigiek megünneplése (és összefoglalása) kedvéért kocsmázzunk egyet!

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

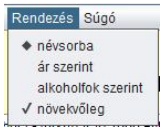
Ahogy látható, az itallapon szereplő italok között vannak alkoholos és nem alkoholos italok is. Az ital definiálásakor meg kell adnunk a fajtáját (bor, tea, víz, stb), vonalkódját és literenkénti árát. Az alkoholos italt a fentieken kívül még egy márkanév és az alkoholfok is jellemzi. Az egyes italokhoz az egyszerűség kedvéért rendeljünk default mennyiségeket.



A nyíllal jelölt gomb hatása: bekerülnek a rendelt italok a választott italok listájába.

A Fizet gomb hatására „fizessen”, törölje ki a választott italok listáját, és írja ki a fizetendő összeget (nyilván a ténylegesen rendelt mennyiségek árát).

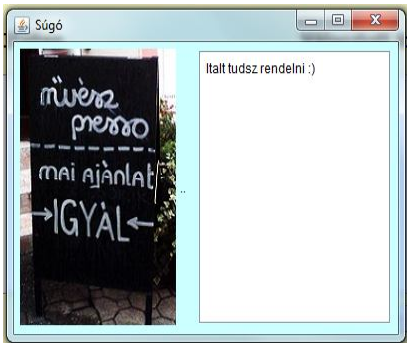
A választott italok listájára kattintva az itallap alatt jelenjen meg a kijelölt ital alkoholfoka, ha van, egyébként pedig az, hogy nem tartalmaz alkoholt.



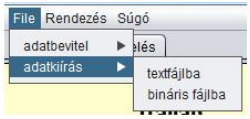
A Rendezés menüpont hatására lehessen rendezni a kiválasztott módon.

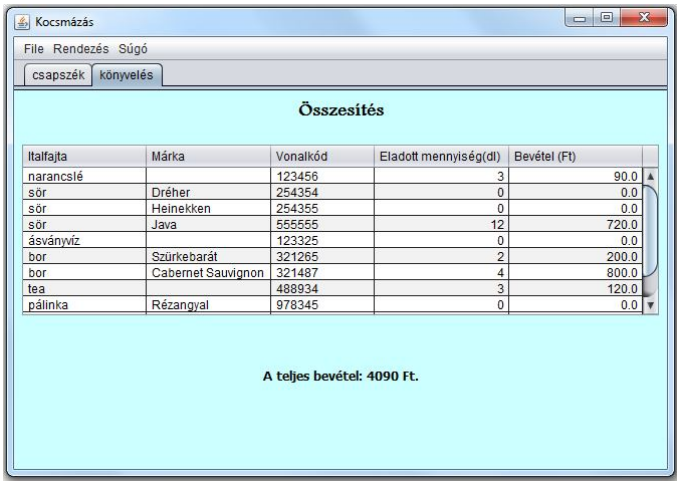
Ha névsorba vagy ár szerint rendezünk, akkor a kiíratás maradjon olyan, amilyennek a képen látja, de ha alkoholfok szerint, akkor az alkoholos italok mellé kerüljön oda zárójelben az alkoholfok értéke is. A menüpont csak a sikeres adatbevitel után váljon aktívvá.

A Súgó menüpont hatása:



A File menüpont másik almenüje:



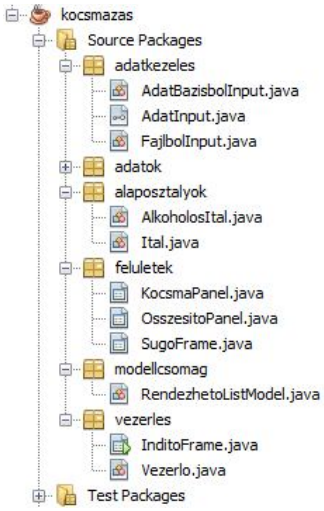


A könyvelés kartotékfül is csak a sikeres beolvasás után váljon aktívvá, és a hozzá tartozó panelen lehessen látni a kocsma kimutatását:

Ne felejtse el tesztelni az alaposztályokat!

Megoldásrészletek:

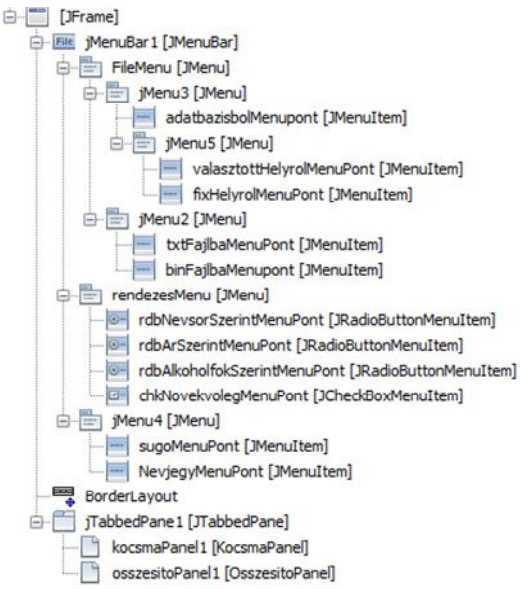
Egy lehetséges projekt-szerkezet:



A feladat megoldását a felület kialakításával érdemes kezdeni. Ebben a korábbiakhoz képest csak a menüsor jelent újdonságot. Ez a frame felületére kerül.

NetBeans fejlesztőkörnyezetet használva design módban rá lehet húzni a felületre. Nyilván hasonlóan egyszerű módon lehet dolgozni vele más fejlesztő környezetben is.

Legjobb, ha önállóan próbálgatja, de az interneten vannak oktató videók is.



A menüsor felrakása, illetve kialakítása után kerülhet a frame felületére a tabulált panel, arra pedig a két panel osztály egy-egy példánya.

Nyilván át kell neveznünk azokat a menüpontokat, amelyekre hivatkozunk a kódból. A bemutatott megoldás során használt elnevezések olvashatóak a frame kialakítását mutató ábrán.

Ahhoz, hogy be tudjuk olvasni az adatokat, és meg tudjuk jeleníteni őket, az alaposztályokból elég a konstruktorokat ismerni. Egyelőre ezeket és a beolvasásokat beszéljük meg, az alaposztályok többi funkcióját majd csak ez után részletezzük.

```java
public class Ital {

    private String fajta;
    private String vonalKod;
    private int literAr;
    private double defaultDeci;

    public Ital(String fajta, String vonalKod, int literAr, double defaultDl) {
        this.fajta = fajta;
        this.vonalKod = vonalKod;
        this.literAr = literAr;
        this.defaultDeci = defaultDl;
    }
```

```java
public class AlkoholosItal extends Ital{

    private String markaNev;
    private double alkoholFok;

    public AlkoholosItal(String fajta, String vonalKod, int literAr,
                         String markaNev, double alkoholFok, double defaultDl) {
        super(fajta, vonalKod, literAr, defaultDl);
        this.markaNev = markaNev;
        this.alkoholFok = alkoholFok;
    }
```

Mivel menüválasztástól függően vagy fájlból, vagy adatbázisból akarunk olvasni, ezért érdemes a beolvasást külön osztályba szervezni, mégpedig úgy, hogy azok egy közös interfészt implementáljanak:

```java
public interface AdatInput {
    public List<Ital> italLista() throws Exception;
}
```

```java
public class AdatBazisbolInput implements AdatInput{

    Connection kapcsolat;

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

    @Override
    public List<Ital> italLista() throws Exception {
        List<Ital> italok = new ArrayList<>();

        String sqlUtasitas = "select * from ITALOK";
```

```
        String fajta, vonalkod, marka;
        int literAr;
        double alkoholFok;
        double defaultDl;

        Ital ital;

        try (Statement utasitasObjektum = kapcsolat.createStatement();
                ResultSet eredmenyHalmaz =
                                    utasitasObjektum.executeQuery(sqlUtasitas)) {
            while (eredmenyHalmaz.next()) {
                fajta = eredmenyHalmaz.getString("fajta");
                vonalkod = eredmenyHalmaz.getString("vonalkod");
                literAr = eredmenyHalmaz.getInt("literar");
                marka = eredmenyHalmaz.getString("marka");
                alkoholFok = eredmenyHalmaz.getDouble("alkoholfok");
                defaultDl = eredmenyHalmaz.getDouble("defaultdl");

                if(marka == null){
                    ital = new Ital(fajta, vonalkod, literAr, defaultDl);
                }
                else ital = new AlkoholosItal(fajta, vonalkod,
                                        literAr, marka, alkoholFok, defaultDl);
                italok.add(ital);
            }
        }
        return italok;
    }
}
```

A fájlból olvasó osztály azért igényel külön magyarázatot, mert kétféle módon is olvashatunk: egyrészt egy fix, megadott címen lévő fájlból, másrészt egy fájlválasztó ablak segítségével kiválasztottból. A probléma az, hogy az első esetben inputstream-en keresztül kell olvasnunk (ekkor lehet relatív módon úgy megadni az elérési útvonalat, hogy .jar fájlból indítva is megtalálja a fájlt), fájlválasztás esetén viszont közvetlenül a fájlból való olvasás a kényelmes. (Természetesen a .jar fájl az adatbázist is eléri, feltéve, ha működik az adatbázisszerver.)

Az említett okok miatt úgy építjük fel az osztályt, hogy mindkét módon tudjon olvasni, vagyis akkor is, ha az adatfájlt ismerjük, és akkor is, ha annak elérési útvonalát.

```java
public class FajlbolInput implements AdatInput {

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

    private enum Honnan {fajlbol, streambol}
    private Honnan honnan;

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

    public FajlbolInput(String italEleres) {
        this.italEleres = italEleres;
        honnan = Honnan.streambol;
    }

    @Override
    public List<Ital> italLista() throws Exception {

        List<Ital> italok = new ArrayList<>();
        Scanner fajlScanner = null;

        switch (honnan) {
            case fajlbol: {
                fajlScanner = new Scanner(adatFajl, CHAR_SET);
                break;
            }
            case streambol: {
                InputStream ins = this.getClass().getResourceAsStream(italEleres);
                fajlScanner = new Scanner(ins, CHAR_SET);
                break;
            }
            default:
                throw new Exception();
        }
        String sor;
        String[] adatok;
        Ital ital = null;
        while (fajlScanner.hasNextLine()) {
            sor = fajlScanner.nextLine();
            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]));
                }
            }
            if (ital != null) {
                italok.add(ital);
            }
        }
        return italok;
    }
}
```

Magára a beolvasásra az egyes menüpontok hatására kerül sor. A frame ide tartozó része:

```
public class InditoFrame extends javax.swing.JFrame {

    private final int SZELESSEG = 666;
    private final int MAGASSAG = 468;
    private final String CIM = "Kocsmázás";

    private Vezerlo vezerlo;

    public InditoFrame() {
        initComponents();
        beallitas();
    }

    private void beallitas() {
        this.setSize(SZELESSEG, MAGASSAG);
        this.setTitle(CIM);
        this.setLocationRelativeTo(this);
        rendezesMenu.setEnabled(false);
        jTabbedPane1.setEnabledAt(1, false);
    }


    private void adatbazisbolMenupontActionPerformed(java.awt.event.ActionEvent
        if (vezerlo.adatBazisbol()) aktivalas();
    }

    private void fixHelyrolMenuPontActionPerformed(java.awt.event.ActionEvent e
        if (vezerlo.fixFajlbol()) aktivalas();
    }

    private void valasztottHelyrolMenuPontActionPerformed(java.awt.event.Actior
        if (vezerlo.valasztottHelyrol()) aktivalas();
    }

    private void aktivalas() {
            rendezesMenu.setEnabled(true);
            jTabbedPane1.setEnabledAt(1, true);
    }
```

Természetesen kell bele egy olyan metódus is, amelyben bemutatjuk egymásnak a vezérlő és a panel osztályokat (a `main()` metódusból hívjuk):

```
    private void start() {
        this.setVisible(true);
        vezerlo = new Vezerlo(kocsmaPanel1, osszesitoPanel1);
        kocsmaPanel1.setVezerlo(vezerlo);
    }
```

A vezérlő osztály megfelelő metódusai:

```
public class Vezerlo {

    private final String ITAL_ELERES = "/adatok/arlista.txt";
    private KocsmaPanel kocsmaPanel;
    private OsszesitoPanel osszesitoPanel;
    private List<Ital> italok = new ArrayList<>();

    public Vezerlo(KocsmaPanel kocsmaPanel, OsszesitoPanel osszesitoPanel) {
        this.kocsmaPanel = kocsmaPanel;
        this.osszesitoPanel = osszesitoPanel;
    }

    public boolean fixFajlbol() {
        try {
            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;
        }
    }

    public boolean valasztottHelyrol() {
        // A választó ablak a projekt gyökerében (aktuális mappa) nyíljon meg.
        JFileChooser fajlValaszto = new JFileChooser(new File("."));
        if (fajlValaszto.showOpenDialog(kocsmaPanel) == JFileChooser.APPROVE_OPTION) {
            try {
                File fajl = fajlValaszto.getSelectedFile();
                AdatInput adatInput
                        = new FajlbolInput(fajl);
                if (adatBevitel(adatInput)) {
                    return true;
                }
            } catch (Exception ex) {
                Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                        null, ex);
            }
        }
        return false;
```

```
private boolean adatBevitel(AdatInput adatInput) throws Exception {
    italok = adatInput.italLista();
    if (!italok.isEmpty()) {
        kocsmaPanel.itallapMegjelenites(Collections.unmodifiableList(italok));
        return true;
    } else {
        JOptionPane.showMessageDialog(kocsmaPanel, "Hibás adatfájl");
        return false;
    }
}

public boolean adatBazisbol() {
    try (Connection kapcsolat = adatBazisKapcsolat()) {
        AdatInput adatInput = new AdatBazisbolInput(kapcsolat);
        adatBevitel(adatInput);
        return true;
    } catch (Exception ex) {
        Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                                                          null, ex);
        return false;
    }
}

private Connection adatBazisKapcsolat() throws ClassNotFoundException,
                                               SQLException {
    // az adatbázis driver meghatározása
    Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
    // az adatbázis definiálása
    String url = "jdbc:derby://localhost:1527/KOCSMA";
    // kapcsolodas az adatbázishoz
    return DriverManager.getConnection(url, "kocsma", "kocsma");
}
```

Megjegyzések:

1. A beolvasó metódusok azért `boolean` típusúak, mert csak sikeres olvasás esetén aktiválódhat az eredetileg inaktív menüpont és kartotékfül.

2. Már volt szó róla, hogy eredeti listát sohasem adunk ki a „kezünkből”, vagyis nem a listát, hanem annak másolatát adjuk át a getterekben. Ha szigorúan és precízen programozunk, akkor ezt az elvet nem csak a getterekben, hanem bármilyen lista-átadáskor követnünk kellene. Ezt a korábbiakban (kényelmi okok miatt) nem tettük meg. Mondhatnánk, hogy a gettert bárki meghívhatja, a panelt viszont mi kérjük meg arra, hogy írassa ki a lista elemeit, ezért megbízhatunk benne, de mégis biztonságosabb, ha nem adjuk át az eredetit. Ennek egyik módja volt az, hogy másolatot adtunk át, itt most egy másik megoldást láthat: becsomagoljuk a listát úgy, hogy ne lehessen módosítani (unmodifiable). (Ekkor az átvevő osztály még csak rendezni sem tudja a listát, míg a másolatot tudja.)

3. A hibaüzenetre főleg a választható fájl miatt van szükség, mégpedig akkor, ha létező, de nem megfelelő fájlt választunk.

Újdonságként majd azt is megmutatjuk, hogy hogyan lehet saját modellt rendelni egy lista-felülethez. Ez olyan modell lesz, amely már rendezni is tud, de majd kicsit később beszélünk róla. Egyelőre azonban dolgozhat a default modellel. A panel adatmegjelenítő metódusa (+ a szükséges deklarációk, beállítások):

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

    private RendezhetoListModel<Ital> italModel = new RendezhetoListModel<>();
    private RendezhetoListModel<Ital> rendeltItalModel = new RendezhetoListModel<>();

    private Vezerlo vezerlo;

    public KocsmaPanel() {
        initComponents();
        lstItallap.setModel(italModel);
        lstValasztottItalok.setModel(rendeltItalModel);
    }

    public void itallapMegjelenites(List<Ital> italok) {
        italModel.clear();
        for (Ital ital : italok) {
            italModel.addElement(ital);
        }
    }

    public void setVezerlo(Vezerlo vezerlo) {
        this.vezerlo = vezerlo;
    }
```

Eljutottunk addig, hogy a `KocsmaPanel` felületén megjelennek a beolvasott adatok (természetesen meg kell írni az ital osztály `toString()` metódusát).

Beszéljük meg a panel eseményeihez tartozó metódusokat.

A választás gomb („nyíl”-gomb – a nyíl külalak a gomb icon tulajdonságaként adható meg, hozzárendelve a nyilat ábrázoló képet) hatására kerülnek át a jobboldali listába a kiválasztott italok. Ez egyáltalán nem tűnik nehéz feladatnak. Csakhogy a látszat csal. Egyrészt azért, mert más a külalakja a két lista elemeinek:

másrészt a logikai funkciójuk is más, hiszen nem ugyanazt az italt kérjük, ami az árlapon van, hanem egy olyat. (Mivel itt inkább csak adminisztrációról van szó, ez utóbbi logikátlanság még beleférne a megoldásba, a fő baj a külalak).

A listafelületen az objektum `toString()` értéke jelenik meg, márpedig nem tartozhat ugyanahhoz az objektumhoz egyszerre két `toString()` érték is. (Azt pedig nyilván nem vizsgálhatjuk, hogy az elem épp melyik listán szerepel, hiszen az alaposztály azt sem tudja, hogy a példányait listába rakják.)

A megoldás az, hogy a kiválasztott példány adatai alapján új példányt hozunk létre, és ez kerül be a másik listába. (Eleve képzelhetjük azt, hogy az itallapon szereplő italok amolyan kis kirakatban mintaként szerepelnek, és nyilván nem ezeket a példányokat szeretnénk választani, hanem ugyanilyen másik példányokat.)

Viszont minden italfajta csak egyszer szerepelhet a listán, és csak egy darabszám mutatja, hogy hányszor választottuk az adott fajtát. Vagyis azt is ellenőrizni kell, hogy van-e már a választottak listájában (modelljében) olyan példány, amilyet épp belerakni készülünk. Az „olyanságot" viszont csak úgy tudjuk ellenőrizni, ha megírjuk az `Ital` osztály `equals()` és `hashCode()` metódusát. Akkor tekintünk ugyanolyannak két példányt, ha azonos a fajtájuk és a vonalkódjuk. (Ezeket a metódusokat egyébként illene mindig generálni, csak most azokra az esetekre szeretnénk koncentrálni, amikor valóban szükség is van rájuk.)

Azt is számlálnunk kell, hogy hány egyetlen rendelés során hány darabot rendeltek az adott italból. Ez a szám lehet a kiíratás kulcsa, ha ugyanis legalább egyet, akkor már ki lehet írni a darabszámot. Ugyanakkor azt is meg kell oldani, hogy a baloldali listában lévő objektumok esetén megmaradjon 0 ez a darabszám, viszont számláljuk, hogy összesen hányszor rendeltek belőle. Ez lesz az osztály `strigulaz()` metódusa. Arra is figyelnünk kell, hogy a fizetés után a rendelt darabszám nyilván ismét nulla lesz, viszont a strigulázott érték továbbra is marad. Ezeket végiggondolva a két alaposztály ide vonatkozó része (getterek, setterek nélkül):

```
public class Ital {

    private String fajta;
    private String vonalKod;
    private int literAr;
    private double defaultDeci;

    private int rendeltDb;
    private int osszDb;

    public Ital(String fajta, String vonalKod, int literAr, double defaultDl) {
        this.fajta = fajta;
        this.vonalKod = vonalKod;
        this.literAr = literAr;
        this.defaultDeci = defaultDl;
    }

    public void rendeles(){
        rendeltDb++;
    }

    public void strigulaz(){
        osszDb++;
    }

    public int rendelesAr(){
        return (int) Math.round(rendeltDb*defaultDeci*literAr/10);
    }
```

```
    public double osszMennyiseg(){
        return osszDb * defaultDeci;
    }

    public double osszBevetel(){
        return (int) Math.round(osszDb*defaultDeci*literAr/10);
    }

    @Override
    public String toString() {
        if(rendeltDb > 0) {
            return String.format("%3d * %3.1f dl %s",
                    rendeltDb, defaultDeci, fajta);
        }
        return fajta + " " + literAr + " Ft/liter";
    }
```

```
public class AlkoholosItal extends Ital{

    private String markaNev;
    private double alkoholFok;

    private static boolean alkoholFokSzerintRendezve;

    public AlkoholosItal(String fajta, String vonalKod, int literAr,
                         String markaNev, double alkoholFok, double defaultDl) {
        super(fajta, vonalKod, literAr, defaultDl);
        this.markaNev = markaNev;
        this.alkoholFok = alkoholFok;
    }

    @Override
    public String toString() {
        if (this.getRendeltDb() > 0) {
            return super.toString() + " (" + markaNev + ")";
        } else {
            String temp = String.format("%s (%s) %4d Ft/liter",
                    this.getFajta(), this.markaNev, this.getLiterAr());
            if (alkoholFokSzerintRendezve) {
                return temp + " (" + alkoholFok + "°)";
            }
            return temp;
        }
    }
```

Ezek után következhetnek a `KocsmaPanel` eseményei:

Választás (nyíl) gomb:

```
    private void btnValasztasActionPerformed(java.awt.event.ActionEvent evt) {
        List<Ital> valasztottak = lstItallap.getSelectedValuesList();
        Ital rendeltItal;
        for (Ital ital : valasztottak) {
            vezerlo.adminisztral(ital);
            rendeltItal = ital instanceof AlkoholosItal
                 ? new AlkoholosItal(ital.getFajta(), ital.getVonalKod(),
                         ital.getLiterAr(), ((AlkoholosItal) ital).getMarkaNev(),
                         ((AlkoholosItal) ital).getAlkoholFok(),
                         ital.getDefaultDl())
                 : new Ital(ital.getFajta(), ital.getVonalKod(),
                                    ital.getLiterAr(), ital.getDefaultDl());
            if (!(rendeltItalModel.contains(rendeltItal))) {
                rendeltItal.rendeles();
                rendeltItalModel.addElement(rendeltItal);
            }else{
                int index = rendeltItalModel.indexOf(rendeltItal);
                rendeltItalModel.getElementAt(index).rendeles();
            }
        }
        lblFizetendo.setText("");
        lstValasztottItalok.repaint();
    }
```

A vezérlő hivatkozott metódusa:

```
    public void adminisztral(Ital valasztottItal) {
        int index = italok.indexOf(valasztottItal);
        italok.get(index).strigulaz();
    }
```

A Fizetés gomb:

```
    private void btnFizetActionPerformed(java.awt.event.ActionEvent evt) {
        int ossz = 0;
        for (int i = 0; i < rendeltItalModel.getSize(); i++) {
            ossz += rendeltItalModel.getElementAt(i).rendelesAr();
        }
        lblFizetendo.setText("Fizetendő: " + ossz + " Ft");
        rendeltItalModel.clear();
        lstValasztottItalok.repaint();
        lstItallap.clearSelection();
    }
```

A választott italok listájára kattintás hatása (ne felejtse el beállítani, hogy a listából csak egyetlen elemet lehessen választani):

```
    private void lstValasztottItalokValueChanged(javax.swing.event.ListSelectionEvent
        Ital valasztottItal = lstValasztottItalok.getSelectedValue();
        if (valasztottItal != null) {
            if (valasztottItal instanceof AlkoholosItal) {
                lblAlkoholFok.setText("A választott ital alkoholfoka: " +
                        ((AlkoholosItal) valasztottItal).getAlkoholFok() + " fok");
            } else {
                lblAlkoholFok.setText("A választott ital nem tartalmaz alkoholt.");
            }
        }
    }
```

Megjegyzés:
A választott ital adatai alapján egyszerűbben is lehet új példányt generálnunk, mégpedig a klónozás segítségével. Ehhez pl. itt talál referenciákat:

https://www.tutorialspoint.com/java/lang/object_clone.htm

https://www.javatpoint.com/object-cloning

Ekkor a `rendeltItal` nevű változó a bonyolultnak tűnő megoldás helyett így kaphat értéket:

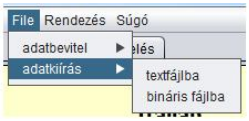
```
            try {
                rendeltItal = (Ital) ital.clone();
                if (!(rendeltItalModel.contains(rendeltItal))) {
                    rendeltItal.rendeles();
                    rendeltItalModel.addElement(rendeltItal);
                } else {
                    int index = rendeltItalModel.indexOf(rendeltItal);
                    rendeltItalModel.getElementAt(index).rendeles();
                }
            } catch (CloneNotSupportedException ex) {
                Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                                                                  null, ex);
            }
```

Ehhez azonban az kell, hogy az Ital osztály implementálja a `Cloneable` interfészt, és megvalósítsa a `clone()` metódust:

```
public class Ital  implements Serializable, Comparable<Ital>, Cloneable {

    @Override
    public Ital clone() throws CloneNotSupportedException{
        return (Ital) super.clone();
    }
```

Következő témánk legyen a fájlba írás.



Elvileg ezt is rakhatnánk olyan osztályokba, amelyek egy adatkiíró interfészt implementálnak, de nem érdemes, hiszen már az is kissé erőltetett, hogy a beolvasást így oldottuk meg, de ez még indokolható a könnyű módosíthatósággal, illetve azzal, hogy láthassa az interfész előnyét. Ha majd később egyéb adatbáziskezelő műveleteket is veszünk, akkor majd a CRUD műveletek leírására lesz célszerű külön interfészt létrehozni.

Most tehát közvetlenül a vezérlésből oldjuk meg a fájlba írást. A szövegfájlba írás egyszerű. Az is egyszerű, ha egy teljes objektumot szeretnénk fájlba írni, ehhez azonban szerializálhatóvá kell tennünk a mentendő objektumot, hiszen csak ilyen objektumot tudunk majd a fájl beolvasásakor visszaalakítani. A `List` interfész szerializálható, de az általunk írt `Ital` osztály még nem. De pillanatok alatt szerializálhatóvá lehet tenni, csak implementálni kell a megfelelő interfészt:

```
public class Ital  implements Serializable{
```

Ezek után a (fájlválasztóval kiválasztott) fájlba írás:

A frame-n:

```
    private void txtFajlbaMenuPontActionPerformed(java.awt.event.ActionEvent evt) {
        vezerlo.textFajlba();
    }

    private void binFajlbaMenupontActionPerformed(java.awt.event.ActionEvent evt) {
        vezerlo.binarisFajlba();
    }
```

A vezérlő osztályban:

```
    private enum HOVA {text, binaris}

    public void textFajlba() {
        fajlbaIr(HOVA.text);
    }

    public void binarisFajlba() {
        fajlbaIr(HOVA.binaris);
    }
```

```
private void fajlbaIr(HOVA hova) {
    JFileChooser fajlValaszto = new JFileChooser(new File("."));
    if (fajlValaszto.showSaveDialog(kocsmaPanel) == JFileChooser.APPROVE_OPTION) {
        File fajl = fajlValaszto.getSelectedFile();
        try {
            if (fajl.exists()) {
                if (JOptionPane.showConfirmDialog(kocsmaPanel, "felülírhatom?")
                                                == JOptionPane.OK_OPTION) {
                    kiir(fajl, hova);
                }
            } else {
                kiir(fajl, hova);
            }
        } catch (IOException ex) {
            Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                                                              null, ex);
        }
    }
}

private void kiir(File fajl, HOVA hova) throws IOException {

    switch (hova) {
        case text: {
            try (PrintWriter ki = new PrintWriter(new FileWriter(fajl))) {
                for (Ital ital : italok) {
                    ki.println(ital.fajlbaString());
                }
            }
            break;
        }
        case binaris: {
            try (FileOutputStream fout = new FileOutputStream(fajl);
                    ObjectOutputStream oout
                    = new ObjectOutputStream(new BufferedOutputStream(fout))) {
                oout.writeObject(italok);
                oout.flush();
            }
        }
        default: throw new IOException();
    }
}
```

A hivatkozott `fajlbaString()` metódus egy olyan stringet ad vissza, amilyennek az eredmény fájl egy-egy sorát szeretnénk. Ha pl. az italjellemzők mellett a forgalmat is meg szeretnénk jeleníteni, akkor ilyen az `Ital` osztályban:

```
public String fajlbaString(){
    return alapString() + ";" + osszMennyiseg() + ";" + osszBevetel();
}

protected String alapString() {
    return fajta + ";" + vonalKod + ";" + literAr + ";" + defaultDeci;
}
```

`AlkoholosItal` osztályban:

```
@Override
protected String alapString() {
    return super.alapString() + ";" + markaNev + ";" + alkoholFok;
}
```

Hátra van a rendezés megbeszélése. Korábbi feladatokban volt már szó arról, hogyan lehet különböző szempontok szerint rendezni (pl, bálozók, gólyabál feladat). Eddigi ismereteink szerint a vezérlő osztályban rendezni kellene az italok listáját, majd átadni a panel osztálynak, mivel a `DefaultListModel` típushoz nincs rendezési funkció rendelve. Azonban mi saját magunk is írhatunk modellt, akár olyat is, amelyik rendezni is képes.

Egy modell többek között abban különbözik pl. egy util listától, hogy a benne végbemenő változásokról azonnal értesíti a hozzá tartozó listafelületet, vagyis „tüzel”. Erre vonatkoznak a különböző `fire...()` metódusok.

A következőkben mutatunk egy lehetséges saját modellt – a benne szereplő metódusok közül csak néhányat használunk ennek a feladatnak a megoldásához. Egy saját modellt az `AbstractListModel` osztály leszármazottjaként deklarálhatunk, amelynek két kötelezően implementálandó metódusa van, a `getElementAt()` és a `getSize()` metódus. A többit tetszőlegesen (de nyilván a céljainknak megfelelően) írhatjuk meg.

Egy lehetséges modell tehát (ha ki akarja próbálni, elég csak a feltétlenül szükséges metódusokat megírni belőle, de le is töltheti a feladatkiírásban megadott helyről):

```
/**
 * Saját, rendezhető listamodell.
 * Bármely Comparable típusra alkalmazható.
 * @param <T>
 */
public class RendezhetoListModel<T extends Comparable<T>>
                        extends AbstractListModel {

    // Ha listában tároljuk a modell elemeit, akkor használhatjuk
    // a lista kényelmes szolgáltatásait,
    // de meg kell írnunk a tüzelő metódusokat.

    private List<T> modell = new ArrayList<>();

    // Rendezett-e a lista
    private boolean rendezett = false;
```

```java
// Kötelező implementálni a köv. két metódust (generálható):
 @Override
 public T getElementAt(int index) {
     return modell.get(index);
 }

 @Override
 public int getSize() {
     return modell.size();
 }

 /**
  * Ha utólag akarjuk rendezni a modellt, akkor ezt
  * a metódust kell hívni.
  * (Látható, hogy használható a lista-rendezés.)
  * A rendezés után jelzi, hogy megváltozott a tartalma,
  * ezt a hozzárendelt JList kezeli majd.
  */
public void sort() {
    Collections.sort(modell);
    this.fireContentsChanged(this, 0, modell.size() - 1);
}


 /**
  * a modell adott indexű helyére szúr be egy elemet,
  * és jelzi, hogy ez meg is történt.
  * @param elem
  * @param index
  */
public void addElement(T elem, int index) {
    modell.add(index, elem);
    this.fireIntervalAdded(this, index, index);
}

/**
 * a modell végére szúr be egy elemet,
 * és jelzi, hogy ez meg is történt.
 * @param elem
 */
public void addElement(T elem) {
    this.addElement(elem, modell.size());
    this.fireIntervalAdded(elem, modell.size()-1, modell.size()-1);
}
```

```java
/**
 * A rendezésre figyelve szúrja be az adott elemet.
 *
 * Ha a rendezve értéke false, akkor a modell végére teszi az új elemet,
 * és jelzi, hogy a modell nem rendezett.
 *
 * Ha a rendezve értéke true, akkor már eleve rendezetten rakja be az elemet.
 * Ha az eddig berakottakat elemek nem rendezetten szerepelnek a modellben,
 * akkor előbb ezeket is rendezi, és az ennek megfelelő helyre rakja az újat.
 * A hely megkereséséhez a bináris rendezést használja.
 *
 * @param element
 * @param rendezve
 */
public void addElement(T element, boolean rendezve) {

    // Ha nem rendezve kérjük, akkor az új elemet a modell végére teszi.
    if (!rendezve) {
        this.addElement(element);
        rendezett = false;
    }
    // Ha rendezve kérjük, akkor
    else {
        // ha a modell eddig nem volt rendezve, akkor most rendezi
        if (!rendezett) {
            sort();
        }
        // Megkeresi az új elem bináris rendezésnek megfelelő helyét.
        int index = Collections.binarySearch(modell, element);


        // A Collections.binarySearch() metódus a bináris keresés alapján
        // megkeresi egy rendezett listában az adott elem indexét.
        // Ha a lista nincs rendezve, akkor a hívás eredménye definiálatlan.
        // Az index megadja a keresett elem indexét, ha létezik ilyen elem
        // a listában, egyébként: (-(beszurasiPont) - 1).
        // A beszurasiPont az a hely, ahova be lehetne szúrni az elemet.
        // Pontosabb részleteket ld. help.

        if (index < 0)
            addElement(element, -index - 1);
        else
            addElement(element, index);
        rendezett = true;
    }
}
```

```
    /**
     * Lekérdezhető, hogy rendezett-e a modell
     * @return true, ha rendezett, false, ha nem
     */
    public boolean isRendezett() {
        return rendezett;
    }

    /**
     * Beállítható, hogy rendezettnek tekintjük-e a modellt.
     * @param rendezett
     */
    public void setRendezett(boolean rendezett) {
        this.rendezett = rendezett;
    }

    public boolean contains(T element){
        return modell.contains(element);
    }


    public void removeElement(T element){
        int index = modell.indexOf(element);
        modell.remove(element);
        this.fireIntervalRemoved(element, index, index);
    }

    public void clear() {
        modell.clear();
    }

    public int indexOf(T element) {
        return modell.indexOf(element);
    }
}
```

A modell elemeinek összehasonlíthatóknak kell lenniük, vagyis az `Ital` osztálynak implementálnia kell a `Comparable` interfészt. Az interfész kötelezően implementálandó metódusa a `compareTo()` metódus, ezt is meg kell valósítani az Ital osztályban. Mivel most többféle rendezési szempont is lehet, ezért – a korábban megbeszélt külön rendező osztályhoz hasonlóan – most ebben a metódusban kell odafigyelnünk a szempontokra.

Az `Ital` osztály ilyen irányú bővítése (getterek nélkül):

```
public class Ital  implements Serializable, Comparable<Ital> {

    public enum Szempont{FAJTA, AR, ALKOHOLFOK}

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

    private static Szempont valasztottSzempont;
    private static boolean miModon;

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

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

Ezek után a rendezés már gyerekjáték.

A frame osztályban:

```
    private void rdbArSzerintMenuPontActionPerformed(java.awt.event.ActionEvent evt) {
        vezerlo.arSzerint(chkNovekvolegMenuPont.isSelected());
    }

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

    private void rdbAlkoholfokSzerintMenuPontActionPerformed(java.awt.event.ActionEvent
        vezerlo.alkoholfokSzerint(chkNovekvolegMenuPont.isSelected());
    }

    private void chkNovekvolegMenuPontActionPerformed(java.awt.event.ActionEvent evt) {
        novekvoCsokkeno();
    }

    private void novekvoCsokkeno() {
        if(rdbAlkoholfokSzerintMenuPont.isSelected())
              vezerlo.alkoholfokSzerint(chkNovekvolegMenuPont.isSelected());
        if(rdbArSzerintMenuPont.isSelected())
              vezerlo.arSzerint(chkNovekvolegMenuPont.isSelected());
        if(rdbNevsorSzerintMenuPont.isSelected())
              vezerlo.nevsorSzerint(chkNovekvolegMenuPont.isSelected());
    }
```

A vezérlő osztályban:

```
    public void arSzerint(boolean novekvo) {
        AlkoholosItal.setAlkoholFokSzerintRendezve(false);
        Ital.setValasztottSzempont(Ital.Szempont.AR, novekvo);
        kocsmaPanel.rendez();
    }

    public void nevsorSzerint(boolean novekvo) {
        AlkoholosItal.setAlkoholFokSzerintRendezve(false);
        Ital.setValasztottSzempont(Ital.Szempont.FAJTA, novekvo);
        kocsmaPanel.rendez();
    }

    public void alkoholfokSzerint(boolean novekvo) {
        AlkoholosItal.setAlkoholFokSzerintRendezve(true);
        Ital.setValasztottSzempont(Ital.Szempont.ALKOHOLFOK, novekvo);
        kocsmaPanel.rendez();
    }
```

A `KocsmaPanel` osztályban:

```
    public void rendez() {
        italModel.sort();
    }
```

A könyvelés megoldása már nem jelent újdonságot.

A frame osztályban:

```
    private void jTabbedPane1MousePressed(java.awt.event.MouseEvent evt) {
        vezerlo.italokTablazatba();
    }
```

A vezérlő osztályban:

```
    public void italokTablazatba() {
        osszesitoPanel.italokTablazatba(Collections.unmodifiableList(italok));
    }
```

Az `OsszesitoPanel` osztály:

A felület kialakítása olyan, ahogyan már megbeszéltük – rá lehet rakni a panel felületére a `jTable` példányt, és kialakítani a hozzá tartozó modell szerkezetét.

A kód (a generált részletek nélkül):

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

    private DefaultTableModel tablaModell ;
    private int bevetel;

    public OsszesitoPanel() {
        initComponents();
        tblOsszesito.setShowGrid(true);
        tablaModell = (DefaultTableModel) tblOsszesito.getModel();
    }

    public void italokTablazatba(List<Ital> italok) {

        // kitöröljük a tábla korábbi sorait
        int n = this.tablaModell.getRowCount();
        for (int i = n - 1; i >= 0; i--) {
            this.tablaModell.removeRow(i);
        }
```

```
        String marka;
        // létrehozzuk az új sorokat
        for (Ital ital : italok) {
            marka = (ital instanceof AlkoholosItal) ?
                                    ((AlkoholosItal) ital).getMarkaNev(): "";
            Object[] tablaSor = {ital.getFajta(), marka, ital.getVonalKod(),
                ital.osszMennyiseg(), ital.osszBevetel()};
            this.tablaModell.addRow(tablaSor);

            // kiszámoljuk a bevételt
            bevetel += ital.osszBevetel();
        }
        lblBevetel.setText("A teljes bevétel: " + bevetel + " Ft.");
    }
```

Megjegyzés: A törlés mindig veszélyes művelet. Ha a 0-s indexű elemtől fölfelé indulva törölnénk, akkor elég hamar indextúlcsordulás adódna, hiszen a ciklus az eredeti listaméret határáig tartana, holott a törlés hatására a tényleges listaméret állandóan csökken. Ezért kezdjük felülről a törlést.

Ejtsünk még néhány szót a súgóról. Egy kicsit is komolyabb programhoz illik súgót készíteni, mert ez egy a felhasználó iránti tisztelet jele is.

Esetünkben ez a súgó egy másik frame. A frame-re felületét könnyen ki lehet alakítani, mondjuk, rákerül egy panel, arra egy olyan label, amelyre képet helyezünk, illetve egy szövegdoboz, de lehet úgy is, hogy a kép egy kisebb panel háttérképe, és ez kerül rá a felületre. A felület kialakítása után már csak az a dolgunk, hogy értelemszerűen hol láthatóvá, hol láthatatlanná tegyük ezt a felületet.

Az `InditoFrame` ide vonatkozó részlete:

```
    private SugoFrame sugofrm = new SugoFrame();

    private void sugoMenuPontActionPerformed(java.awt.event.ActionEvent evt) {
        sugofrm.setLocationRelativeTo(this);
        sugofrm.setVisible(true);
    }
```

A `SugoFrame` kódja (generált részek nélkül):

```
public class SugoFrame extends javax.swing.JFrame {

    private final int SZELESSEG = 400;
    private final int MAGASSAG = 300;
    private final String CIM = "Súgó";

    public SugoFrame() {
        initComponents();
        beallitas();
    }

    private void beallitas() {
        this.setSize(SZELESSEG, MAGASSAG);
        this.setTitle(CIM);
        txtSugo.setText("Italt tudsz rendelni :)");
        this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    }
```

Fontos megjegyzés: Mint minden frame generálásakor, a `SugoFrame` osztályban is létrejön a `main()` metódus. Nem szerencsés azonban, ha egy projektben több `main()` metódus is van. Persze, ha nem ezt indítjuk, akkor látszólag nincs semmi baj, de nem jó ötlet meghagyni annak lehetőségét, hogy a projekt indításakor a súgó jelenjen meg (hacsak nem pont ez a cél, de most nem ez). Ezért töröljük ki a `SugoFrame` osztályból a `main()` metódust.

Ne felejtsen el teszteket is készíteni legalább az alaposztályokhoz.