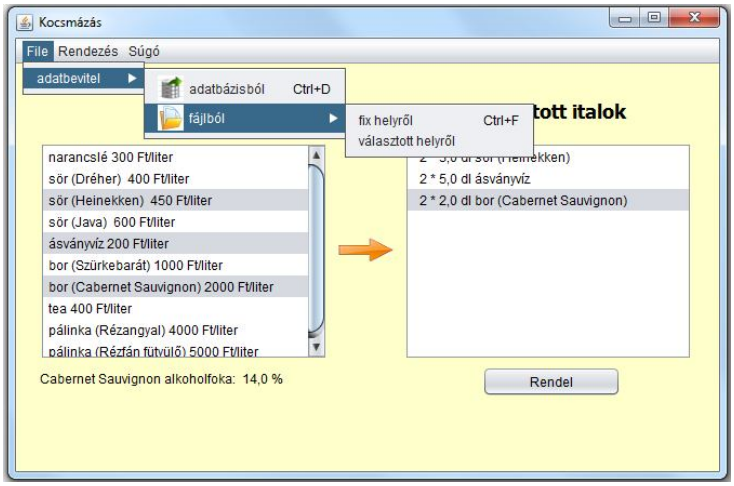
Gyak5 megoldásrészletek

Feladat:

Ha már ennyire bevált a kocsmázás, folytassuk ☺. Módosítsuk a múltkori projektet:



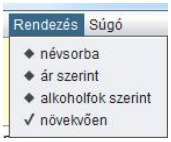
A felület legyen 50 pixellel szélesebb, mint múltkor, és dönthessünk, hogy honnan vesszük az adatokat.

Most engedjük meg, hogy az itallapról egyszerre egynél több italt is választhassunk (ekkor az alkoholfok kiíratását célszerű a másik listához rendelni), és a kiválasztott italok a nyíl gomb megnyomásakor kerüljenek be a választott italok listájába.

Az egyszerűség kedvéért most minden italhoz hozzárendelünk egy default rendelhető mennyiséget, a rendelések száma és ez a mennyiség jelenik meg a jobboldali listafelületen.

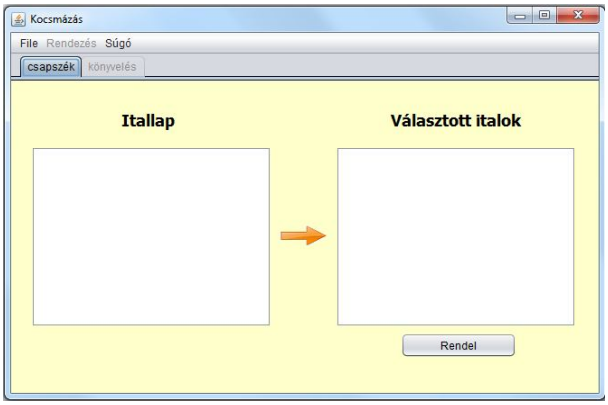
A Rendel feliratú gomb hatására realizálódik a rendelés, ekkor a választott italok eltűnnek a listafelületről, és megjelenik a fizetendő érték.

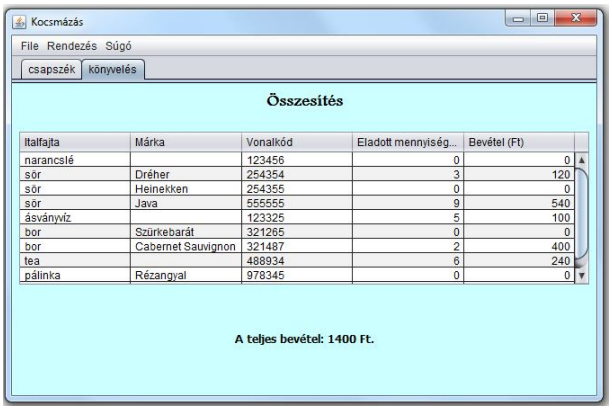
Tesztelje a módosított Ital osztályt!



És hogy ne menjen kárba a múltkor kikínlódott rendezés, a menüpont segítségével rendezzük is az adatokat.

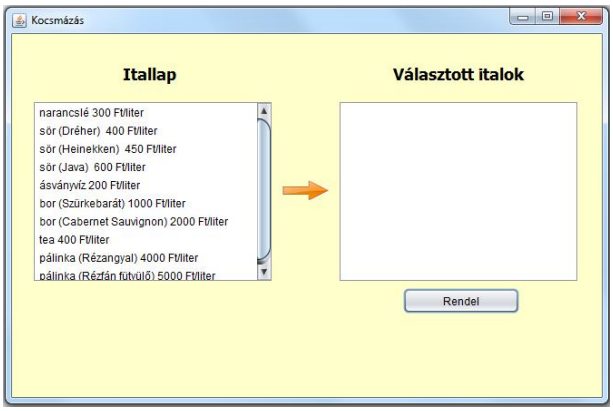
Ha múltkor már megoldotta, akkor most pár perces munka ez az átalakítás:





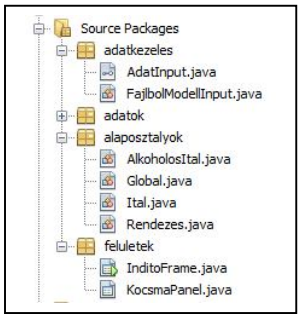
Megoldásrészletek:

A közölt részletek figyelembe veszik a kiadott induló projektet, és innen folytatom, vagyis innen:



A projekt adott helyen lévő fájlból beolvassa az italok adatait (sima ital és alkoholos ital) és megjeleníti a baloldali listafelületen.

A projektben lévő osztályok:



Az alaposztályok módosítása

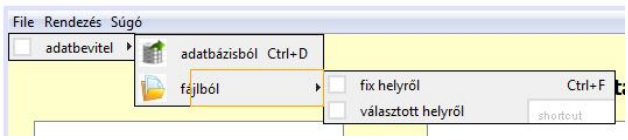
A feladat szerint most minden italhoz hozzárendelünk egy default rendelhető mennyiséget, ezért mindkét alaposztályhoz hozzáadunk egy-egy újabb konstruktort, plusz természetesen a gettereket:

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

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

Adatbevitel

Folytassuk a különböző adatbevitelek megbeszélésével.

Rakjuk fel a frame-re a menü-sort a megfelelő menükkel és menüpontokkal (menu-item):



Az egyes menüpontokból pedig meghívjuk a `KocsmaPanel` osztály megfelelő beolvasó metódusait.

Mivel később még ki szeretnék térni a rendezésre is, kicsit foglalkozzunk a Rendezés menüvel is. Ez legyen inaktív addig, amíg nincsenek adatok (a frame konstruktorában állíthatjuk be vagy a menü tulajdonságai között). A beolvasás után válik aktívvá:

```
private void adatBazisbolMenuItemActionPerformed(java.awt.event.ActionEvent evt)
    kocsmaPanel1.adatBazisbol();
    rendezesMenu.setEnabled(true);
}

private void fixFajlbolMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    kocsmaPanel1.fixFajlbol();
    rendezesMenu.setEnabled(true);
}

private void valasztottFajlbolMenuItemActionPerformed(java.awt.event.ActionEvent
    kocsmaPanel1.valasztottFajlbol();
    rendezesMenu.setEnabled(true);
}
```

Kezdjük az adatábisból való olvasással! Ne felejtse el kitörölni a frame `start()` metódusát, és a main() metódusban visszaállítani ezt: `new InditoFrame().setVisible(true);`

Adatbázisból

Ehhez – ahogy a feladathoz írt segítségben is szerepel – előbb létre kell hozni a kapcsolatot, majd a felépült kapcsolattól kérni egy utasításobjektumot, amelyen keresztül végrehajthatunk az adatbázison egy SQL utasítást, és visszakapjuk az eredményhalmazt.

A kapcsolat kialakításáért felelős metódust megírhatjuk a beolvasást végző osztályban is, de ha már van egy olyan osztályunk (`Global`), amelybe kigyűjtöttük a konstans, illetve könnyen módosíthatónak szánt adatokat, akkor célszerű ezt is ott megírni, hiszen ez a metódus is „fix” adatokat tartalmaz, olyanokat, amelyeket később esetleg módosítani akarunk. (Ha kicseréljük az adatbázisszervert, akkor a programnak csak ezt a részét kell átírnunk a megfelelő beállításokra.)

Mivel a Global osztályba célszerű statikus mezőket és metódusokat írni, ezért most ez is statikus lesz (ha a beolvasással azonos osztályban írjuk meg, akkor nem muszáj statikusnak lennie).

```
public static Connection kapcsolat() throws ClassNotFoundException,
                                            SQLException {
    // az adatbázis driver meghatározása
    Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
    // az adatbázis definiálása
    String url = "jdbc:derby://localhost:1527/PROG3";
    // kapcsolodas az adatbázishoz
    return DriverManager.getConnection(url, "kocsma", "kocsma");
}
```

Ha esetleg később át szeretnénk tenni ezt az adatbázist mondjuk egy MySQL szerverre, akkor a programunk változatlanul használható lesz, ha ebben a metódusban kicseréljük a driver-t a megfelelő MySQL driver-re, illetve értelemszerűen módosítjuk az adatbázis elérési módját. (És természetesen elérhetővé tesszük a driver kezeléséhez szükséges osztályokat.)

A beolvasást végző osztály:

```
public class AdatBazisbolModellInput implements AdatInput{
```

A megfelelő metódus:

```
/**
 * Egy italmodellt visszaadó metódus - az adatokat adatbázisból veszi.
 *
 * @return
 * @throws Exception
 */
@Override
public DefaultListModel<Ital> italModell() throws Exception {
    DefaultListModel<Ital> italModell = new DefaultListModel<>();
        String sqlUtasitas = "select * from ITALOK";
        String fajta, kod, marka;
        int literAr;
        double alkoholFok, defaultDeci;
        Ital ital;

    // Létrehozzuk a kapcsolatot, elkérjük tőle az utasításobjektumot
    // ez végrehajtatja az sql utasítást, és a kapott eredmény bekerül
    // az eredményhalmazba.
    // Mivel mindegyik lezárandó objektum, ezért célszerű a try blokk
    // fejében megnyitni őket.
    try (Connection kapcsolat = Global.kapcsolat();
            Statement utasitasObjektum = kapcsolat.createStatement();
                ResultSet eredmenyHalmaz =
                        utasitasObjektum.executeQuery(sqlUtasitas)) {

        // végigiterálunk a kapott eredményhalmazon
        while(eredmenyHalmaz.next()){
            fajta = eredmenyHalmaz.getString("fajta");
            kod = eredmenyHalmaz.getString("vonalkod");
            literAr = eredmenyHalmaz.getInt("literar");
            marka = eredmenyHalmaz.getString("marka");
            alkoholFok = eredmenyHalmaz.getDouble("alkoholfok");
            defaultDeci = eredmenyHalmaz.getDouble("defaultdl");

            if(marka == null){
                ital = new Ital(fajta, kod, literAr, defaultDeci);
            }else{
                ital = new AlkoholosItal(fajta, kod, literAr, marka,
                                         alkoholFok, defaultDeci);
            }
            italModell.addElement(ital);
        }
    }
    return italModell;
}
```

Ezt a metódust kell meghívnunk az adatbázisból való olvasáskor – erre kicsit később térek vissza.

Ha készen lenne az adatbázisunk, és elérhetővé lenne téve a driver működéséhez szükséges osztályok csomagjainak gyűjteménye, akkor már futtathatnánk is. Ha most nem akar foglalkozni az adatbázis létrehozásával, akkor az órán létrehozottat is használhatja. Ehhez a Netbeans-ben is be kell állítani az adatbázis elérését (ha külső szerveren lenne, akkor erre nyilván nem lenne szükség, de most „embedded”, vagyis a programba beágyazott lokális adatbázist használunk). Ezt

így teheti meg: Services fül, Databases/JavaDB, ezen jobb egérgomb, Properties, és itt megadja a saját adatbázisának elérését. Majd Start Server. Ezek után adatbázisnév, jobb egérgomb, Connect. Először valószínűleg kérni fogja a user-password párost.

Ha még nincs adatbázis, vagy ismét ki akarja próbálni, hogy hogyan is lehet létrehozni, akkor csinálja végig a feladatsorban leírt lépéseket.

Az adatbázisból való olvasást a `KocsmaPanel` osztályban hívjuk meg:

```
void adatBazisbol() {
    AdatInput adatInput = new AdatBazisbolModellInput();
    adatBevitel(adatInput);
}

void adatBevitel(AdatInput adatInput){
    try {
        italModell = adatInput.italModell();
        lstItallap.setModel(italModell);
    } catch (Exception ex) {
        Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Vegye észre az interfészként való deklarálás előnyét: mindhárom adatbeviteli mód esetén ugyanazt az `adatBevitel()` metódust hívjuk meg, kizárólag az változik, hogy hogyan (melyik implementáló osztály alapján) példányosítjuk a deklarált `adatInput` példányt.

Fájlból való olvasás:

A fix helyen lévő fájlból való olvasáshoz lényegileg semmit sem kell változtatni az induló projekthez képest:

```
void fixFajlbol() {
    AdatInput adatInput = new FajlbolModellInput(Global.ADAT_ELERES);
    adatBevitel(adatInput);
}
```

Ha a program futása közben választjuk ki az adatfájlt, akkor viszont kényelmesebb lenne, ha magát az adatfájlt adhatnánk át a beolvasó osztálynak, és nem kellene nehézkes módon kihámozni a fájl elérési útját (az abszolút útvonalat könnyű lekérdezni, de az eredeti adatelérés relatív útvonalat vár). Ezt könnyedén meg is tehetjük, ha a beolvasás során nem az inputstream-en, hanem közvetlenül a fájlon keresztül olvasunk, vagyis ilyen módon érnénk el az adatokat, azaz közvetlenül a fájlt nyitnánk meg a beolvasáskor (`italModell()` metódus):

```
//  try (InputStream ins = this.getClass().getResourceAsStream(adatEleres);
//          Scanner fajlScanner = new Scanner(ins, Global.CHAR_SET)) {
    try (Scanner fajlScanner = new Scanner(adatFajl, Global.CHAR_SET)) {
```

(A kommentezett rész volt az eredeti, ennek helyére kerül az új elérés.)
Ehhez természetesen az osztály konstruktorát is módosítanunk kell:

```
public FajlbolModellInput(File adatFajl) {
    this.adatFajl = adatFajl;
}
```

Dolgozhatnánk úgy, hogy ez egy másik, az `AdatInput` interfészt implementáló osztály lenne – természetesen más néven (a médiás feladat megoldásában így szerepel), de az is megoldható, hogy ezt az osztályt példányosítjuk a fix helyről való olvasáskor is. Fix adatelérés esetén is megadható az adatfájl, csak (szerintem) kicsit nehézkesebb módon, mint ahogy az inputstream-et adtuk meg. (De csak picit nehézkesebb.) Természetesen az Ön döntése, hogy melyik megoldást választja.

A fix helyről való olvasás módosított változata:

```
void fixFajlbol() {
    try {
        File adatFajl = new File(this.getClass().
                                 getResource(Global.ADAT_ELERES).toURI());
        AdatInput adatInput = new FajlbolModellInput(adatFajl);
        adatBevitel(adatInput);
    } catch (URISyntaxException ex) {
        Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

(A `toURI()` metódus megköveteli a kivételkezelést.)

Ezek után a választott fájlból való olvasás nagyon egyszerű. Definiálni kell egy fájlválasztót. Szerintem kényelmesebb kódból megadni:

```
private JFileChooser fajlValaszto;

public KocsmaPanel() {
    initComponents();
    fajlValaszto = new JFileChooser(new File("."));
}
```

Az olvasás:

```
void valasztottFajlbol() {
    if (fajlValaszto.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
        File adatFajl = fajlValaszto.getSelectedFile();
        AdatInput adatInput = new FajlbolModellInput(adatFajl);
        adatBevitel(adatInput);
    }
}
```

Rendelés

Ahhoz, hogy meg tudjuk oldani a feladatban kért rendelést, ismét módosítanunk kell az ital osztályokat. Egyrészt azért, hogy tudja számlálni a megrendeléseket, másrészt azért, mert különböző formátumban szeretnénk megjeleníteni őket a listafelületen, ezért át kell írnunk a `toString()` metódusukat.

Feltételezem, hogy magyarázat nélkül is megérti a kódrészleteket.

`Ital` osztályban:

```
public int defaultAdagAr(){
    return (int) (this.literAr * defaultDeci /10);
}

public void rendeles(){
    rendeltDb++;
}

public int defaultBevetel(){
    return rendeltDb*defaultAdagAr();
}

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

Plusz természetesen a szükséges getterek.

`AlkoholosItal` osztályban:

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

A panel „választás” (nyíl) gombjának hatására a kijelölt italokból rendelnünk kell, és be kell raknunk őket a másik listafelülethez tartozó modellbe. Figyelni kell rá, hogy mindegyik csak egyszer kerüljön be. Ha látni akarjuk a darabszámot, akkor frissítenünk kell a felületet (meg kell hívni a listafelület `repaint()` metódusát.) Látszólag jól is működik, egészen addig, amíg kizárólag ugyanazokat a kiválasztott italokat rendeljük meg. De abban a pillanatban, mihelyt másikat választunk, a baloldali listafelület is frissül, és mivel már van rendelés, a rendelt italok mellé is a darabszám kerül.

A megoldás az, hogy nem ezeket a példányokat rakjuk át, hanem újakat készítünk. (Ez egyébként logikus is, hiszen a kocsmában sem az itallaphoz esetleg mintaként hozzárendelt ital példányokat kérjük, hanem ugyanolyan italokat.)

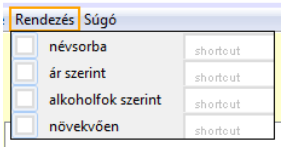
Ezt végiggondolva a gombnyomáshoz tartozó metódus:

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

    Ital aktualisItal;
    int index;
    for (Ital ital : valasztottItalok) {

        if (ital instanceof AlkoholosItal) {
            aktualisItal = new AlkoholosItal(ital.getFajta(),
                                             ital.getVonalKod(),
                                             ital.getLiterAr(),
                                             ((AlkoholosItal) ital).getMarkaNev(),
                                             ((AlkoholosItal) ital).getAlkoholFok(),
                                             ital.getDefaultDeci());
        } else {
            aktualisItal = new Ital(ital.getFajta(), ital.getVonalKod(),
                    ital.getLiterAr(), ital.getDefaultDeci());
        }
        if (valasztottItalModell.contains(aktualisItal)) {
            index = valasztottItalModell.indexOf(aktualisItal);
            aktualisItal = valasztottItalModell.get(index);
        }
        aktualisItal.rendeles();
        if (!valasztottItalModell.contains(aktualisItal)) {
            valasztottItalModell.addElement(aktualisItal);
        }
    }
    // mindkettő jó, de csak az egyik kell - this a panel
    lstValasztottItalok.repaint();
    this.repaint();
}
```

FIGYELEM! Mivel most nem azonos, hanem ugyanolyan példányokat figyelünk, ezért a `contains()` metódus csak akkor működik, ha megírtuk (generáltuk) az `Ital` és az `AlkoholosItal` osztályban is az `equals()` + `hashcode()` metódusokat. Figyeljen rá, hogy a `rendeltDb` NE kerüljön be az egyformaság feltételei közé!

Rendezések

Mivel az induló projektben már megvolt a korábban megírt rendezés, most nagyon kevés dolgunk van.
Egyrészt be kell szúrni a menüsorba a Rendezés menüpontot. (Az első három pont rádiógomb, a negyedik checkbox.)



Összesen annyit kell változtatnunk, hogy most nem a panelen lévő rádiógombokat figyeljük, hanem a menüpontban lévőket. Ezek azonban egy másik osztályban vannak (frame), illetve a

rendezés módját egy checkbox állítja be, ugyancsak a frame-n. A checkbox állapotát egy boolean változóban át kell adnunk a panelnek, a checkbox-váltás kezelését viszont egyszerűbb megoldani a frame osztályban.

Az `InditoFrame` osztályban:

```
private void rdbNevsorbaMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    kocsmaPanel1.nevsorba(chkbNovekoMenuItem.isSelected());
}

private void rdbArSzerintMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    kocsmaPanel1.arSzerint(chkbNovekoMenuItem.isSelected());
}

private void rdbAlkFokSzerintMenuItemActionPerformed(java.awt.event.ActionEvent evt)
    kocsmaPanel1.alkoholFokSzerint(chkbNovekoMenuItem.isSelected());
}

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

private void novekvoCsokkeno() {
    if(rdbNevsorbaMenuItem.isSelected())
        kocsmaPanel1.nevsorba(chkbNovekoMenuItem.isSelected());
    if(rdbArSzerintMenuItem.isSelected())
        kocsmaPanel1.arSzerint(chkbNovekoMenuItem.isSelected());
    if(rdbAlkFokSzerintMenuItem.isSelected())
        kocsmaPanel1.alkoholFokSzerint(chkbNovekoMenuItem.isSelected());
}
```

A `KocsmaPanel` osztályban:

```
public void nevsorba(boolean novekvo) {
    Rendezes.setRendezesiSzempont(Rendezes.Szempont.FAJTA, novekvo);
    rendez();
}

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

public void alkoholFokSzerint(boolean novekvo) {
    Rendezes.setRendezesiSzempont(Rendezes.Szempont.ALKOHOLFOK, novekvo);
    rendez();
}
```