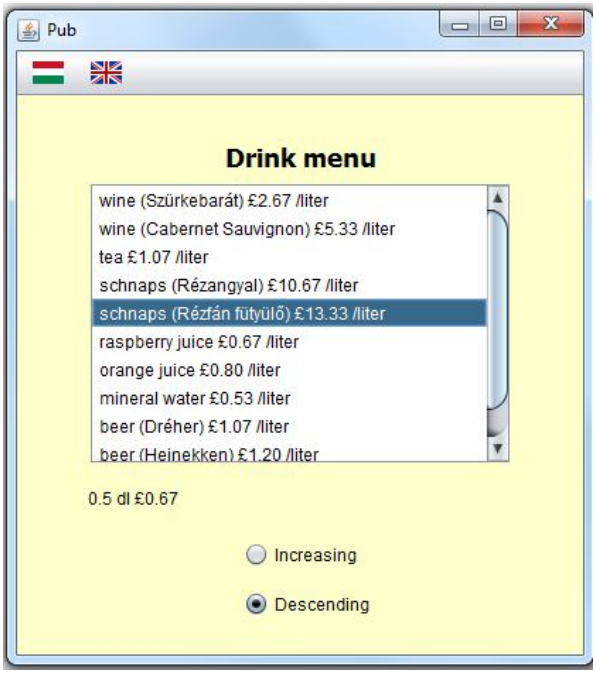
Ásványvíz a helyére! – avagy lokalizáció

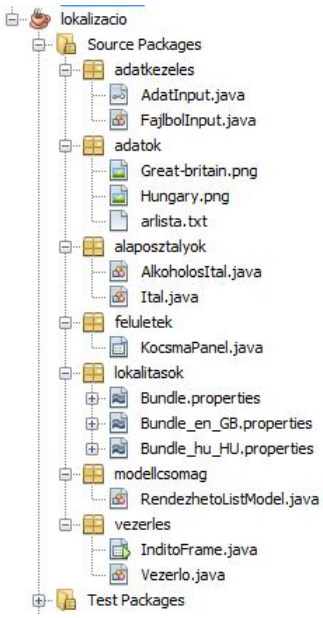
Mint látható volt a kocsmázós feladat megoldása során, a névsorba rendezés nem tökéletes, hiszen az ASCII kódolás miatt az ékezetes karakterek az ábécé végére kerültek, így például az ásványvíz is rossz helyen szerepel. Javítsuk ki ezt a hibát egy, a kocsmázós feladat egyszerűsített változatában!

Most csak annyi a feladat, hogy az italok jelenjenek meg a listafelületen, lehessen őket névsorba rendezni növekvő vagy csökkenő módon, illetve egyetlen kijelölt ital esetén az is jelenjen meg a lista alatt, hogy az illető italból mekkora a default mennyiség, illetve mennyi ennek az ára. Viszont ezek az adatok választástól függően vagy magyarul vagy angolul jelenjenek meg:



Mint látható, nem csak a szövegek jelennek meg az adott nyelven, de a számformátum (tizedesvessző vagy tizedespont) és a pénzformátum is. Sőt, még az árakban is tükröződik a Font és Ft átváltása.

Megoldásrészletek:



Amit lehet, meghagyunk a kocsmás feladat megoldásából, de a mostani feladathoz szükségtelen részletek jó részét elhagyjuk.

Az adatbeviteli osztály változatlan – az egyszerűség kedvéért most csak a fájlból való olvasást vesszük.

A menükialakítás egyszerű: a két menüponthoz csak ikont rendelünk, szöveget nem (legföljebb egy-egy szóközt).

A modellcsomagban szereplő `RendezhetoListaModel` osztály is ugyanaz, mint a kocsmás feladat esetében.

Az `Ital` osztályból kihagyjuk a rendeléshez és könyveléshez szükséges metódusokat, viszont bővítjük néhány, a kétnyelvűséghez szükséges mezővel és metódussal. Bár most csak névsor szerint akarunk rendezni, de a rendezés többi részét is változatlanul hagyjuk. Lehetőséget adunk a nyelv beállítására (erre szolgál a `Locale` típusú `lokalitas` változó), illetve arra, hogy a fajtanevek összehasonlításakor is figyelembe vegyük az aktuális nyelvet. Erre szolgál a `Collator` típusú `szovegEgyezteto` nevű változó.

Az `Ital` osztály (a „szokásos” setterek, getterek nélkül):

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

    private String fajtaMagyar;
    private String fajtaAngol;
    private String vonalKod;
    private int literArMagyar;
    private double defaultDeci;

    // Ennyi Ft egy Font.
    private static double valtoszamFontFt;

    // Induló értékként beállítjuk a default lokalitást.
    private static Locale lokalitas = Locale.getDefault();

    // Beállítjuk a szövegegyeztetőt
    private static Collator szovegEgyezteto = Collator.getInstance();

    public Ital(String fajtaMagyar, String fajta_Agnol, String vonalKod,
                int magyarLiterAr, double defaultDeci) {
        this.fajtaMagyar = fajtaMagyar;
        this.fajtaAngol = fajta_Agnol;
        this.vonalKod = vonalKod;
        this.literArMagyar = magyarLiterAr;
        this.defaultDeci = defaultDeci;
    }
```

```
protected String fajta(){
   return ("hu".equals(lokalitas.getLanguage())) ? fajtaMagyar : fajtaAngol;
}

protected double literAr(){
   return ("hu".equals(lokalitas.getLanguage())) ? literArMagyar
           : literArMagyar/valtoszamFontFt;
}

public double rendelesAr(){
    return defaultDeci*literAr()/10;
}

@Override
public String toString() {
    NumberFormat penzFormatum = NumberFormat.getCurrencyInstance(lokalitas);
    String penzSzoveg = penzFormatum.format(literAr());
    return fajta() + " " + penzSzoveg + " /liter";
}

public static void setLokalitas(Locale lokalitas) {
    Ital.lokalitas = lokalitas;
    Ital.szovegEgyezteto = Collator.getInstance(lokalitas);
}
```

Mint látható, az ital fajtájának elnevezése most az aktuális nyelv függvénye, hasonlóan a literenkénti ár is. A `toString()` metódusban szintén a nyelvtől függő módon adjuk vissza az ital literenkénti árát.

A lokalitás beállításakor arra is figyelnünk kell, hogy hozzáigazítsuk a szövegegyeztetőt is.

Az `equals()`, `hashcode()` metódusokra most közvetlenül nincs szükség, de kár kitörölni. Arra viszont figyelni kell, hogy most akkor tekintünk egyformának két italt, ha a vonalkódjuk és a `fajta()`-juk azonos.

A rendezést is meghagyjuk az eredeti komplexitásában, de kiemeljük a fajtanevek szerint való rendezést, hiszen pont itt van az egyik lényeges elem: a nyelv figyelembevételével rendezzük a neveket.

A `compareTo()` metódus ide vonatkozó részlete:

```
@Override
public int compareTo(Ital o2) {
    switch (valasztottSzempont) {
        case FAJTA:
            return miModon
                    ? szovegEgyezteto.compare(this.fajta(),o2.fajta())
                    : szovegEgyezteto.compare(o2.fajta(),this.fajta());
```

Az `AlkoholosItal` osztály (setter/getter nélkül):

```
public class AlkoholosItal extends Ital{

    private String markaNev;
    private double alkoholFok;

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

    @Override
    public String toString() {
        NumberFormat penzFormatum =
                NumberFormat.getCurrencyInstance(Ital.getLokalitas());
        String penzSzoveg = penzFormatum.format(this.literAr());

        return String.format("%s (%s) %s /liter",
                this.fajta(), this.markaNev, penzSzoveg);
    }
```

Mivel megváltozott az alaposztályok konstruktora, ezért nyilván a beolvasást is módosítani kell egy kicsit, ezt oldja meg önállóan.

Láthatjuk tehát, hogy az adatok különböző nyelveken való megjelenítése nem túl bonyolult dolog: az adatfájlban minden nyelven megadjuk az adat nyelv-függő részét, számolásokhoz beállítunk néhány váltószámot (esetünkben a valutaárfolyamot), és a nyelv megadásától függően máris lehet váltani a megjelenítendő szöveget, illetve a nyelvtől (vagy a megadott országtól függő) formátumokat – pl. pénznem, tizedespont/tizedesvessző. A `Collator` típusú változó segítségével pedig nyelvfüggő módon tudjuk rendezni ezeket az adatokat.

De mi a helyzet a felületre kerülő konstans szövegekkel (vagy más konstans szöveggel)? Ezek megadásához is a fenti ötlet használható, vagyis külön fájlban (vagy fájlokban) meg kell adni az illető szövegek nyelv-függő változatát. A különbség csak az, hogy nem külön adatfájlba, hanem a projektbe beépülő, úgynevezett *properties* fájlokba írjuk őket, mégpedig kulcs-érték párok alakjában, a programból pedig ezekre a fájlokra hivatkozunk, és a kulcsok alapján kerülnek be az egyes szövegek a programba (vagy a felületre). Ezeket a fájlokat „batyuba” rakjuk („bundle”), és egy `ResourceBundle` típusú változón keresztül tudjuk elérni őket.

A tulajdonságfájlok között van egy kiemelt jelentőségű – esetünkben ez a *Bundle.properties* nevű fájl (nyilván mindegy, hogy mi a neve, de kötelezően .properties kiterjesztésű). Ide kerülnek az alapértelmezetten használt elnevezések – ezek jelennek meg akkor, ha semmit nem mondunk a lokalitásról, vagy ha olyan lokalitást (nyelvet) adunk meg, amelyhez nem írtunk tulajdonságfájlt. Alapértelmezetten az operációs rendszer nyelve a default lokalitás, ha készült ehhez a nyelvhez tulajdonság fájl, akkor alapértelmezetten is az jelenik meg.

Az egyes nyelvekhez (vagy nyelv-ország párokhoz) külön tulajdonságfájlt kell írni. Ezeknek ugyanúgy kezdődik a neve, mint az alapértelmezett tulajdonságfájlnak, csak folytatódnak a megfelelő nyelv, vagy nyelv-ország jelekkel. Esetünkben: *Bundle_en_GB.properties*, illetve *Bundle_hu_HU.properties*.

A fájlok tartalma látható az ábrán. (Most szándékosan más frame-cím szerepel az alapértelmezett fájlban, hogy láthassuk a különbséget – esetleg kipróbálhatja úgy, hogy a tulajdonságfájlok közül hiányozzon az, amelyik az operációs rendszer nyelvéhez tartozik).

NetBeans fejlesztőeszközzel így lehet tulajdonságfájlt létrehozni: csomagnév, jobb egérgomb, new / Other / Properties File. De nyilván más fejlesztőeszköz segítségével is egyszerűen lehet létrehozni.

A három tulajdonságfájl tartalma (Bundle, Bundle_en_GB, Bundle_hu_HU)

FRAME_CIM = Kocsmázás ITALLAP_CIM = Itallap CSOKKENOEN = Csökkenően NOVEKVOEN = Növekvően	FRAME_CIM = Pub ITALLAP_CIM = Drink menu CSOKKENOEN = Descending NOVEKVOEN = Increasing	FRAME_CIM = Kocsma ITALLAP_CIM = Itallap CSOKKENOEN = Csökkenően NOVEKVOEN = Növekvően

NetBeans esetén bármelyik properties fájl nevén jobb egérgomb + Open menüpont, akkor együtt is kezelhető a három fájl:

Key	default language	hu_HU - magyar (Magyarország)	en_GB - English (United Kingdom)
FRAME_CIM	Kocsmázás	Kocsma	Pub
ITALLAP_CIM	Itallap	Itallap	Drink menu
CSOKKENOEN	Csökkenően	Csökkenően	Descending
NOVEKVOEN	Növekvően	Növekvően	Increasing

Ezek után a frame címe pl. így állítható be:

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

    private final int SZELESSEG = 416;
    private final int MAGASSAG = 465;
    private final String BUNDLE = "lokalitasok/Bundle";
    private ResourceBundle bundle ;
    private Locale lokalitas;

    private Vezerlo vezerlo;

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

    private void beallitas() {
        bundle = ResourceBundle.getBundle(BUNDLE);
        this.setSize(SZELESSEG, MAGASSAG);
        this.setTitle(bundle.getString("FRAME_CIM"));
        this.setLocationRelativeTo(this);
    }
```

A frame `start()` metódusát továbbra is a `main()` metódusból indítjuk:

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

(Mivel most nem „szól vissza” a panel a vezérlőnek, ezért arra nincs is szükség, hogy a panel is tudja, hogy ki a vezérlő.)

A nyelvet a frame menüsorában állítjuk be:

```
    private void angolMenuMousePressed(java.awt.event.MouseEvent evt) {
        lokalitas = new Locale("en", "GB");
        nyelvBeallitas();
    }

    private void magyarMenuMousePressed(java.awt.event.MouseEvent evt) {
        lokalitas = new Locale("hu", "HU");
        nyelvBeallitas();
    }

    /** A beállított nyelvet és a bundle-t átadjuk a vezérlésnek ...3 lines */
    private void nyelvBeallitas() {
        bundle = ResourceBundle.getBundle(BUNDLE, lokalitas);
        vezerlo.lokalitasEsBundleBeallitas(lokalitas, bundle);
        this.setTitle(bundle.getString("FRAME_CIM"));
    }
```

A vezérlő osztálynak most nem sok dolga van: be kell olvasnia az adatokat, kérni a panelt, hogy jelenítse meg, illetve még az, hogy továbbítsa a panel számára a nyelvi beállításokat.

A Font – Ft árfolyamot most konstansként adtuk meg, de nyilván éles feladat esetén be kellene olvasnunk.

```
public class Vezerlo {

    private final String ITAL_ELERES = "/adatok/arlista.txt";
    private final int ARFOLYAM_FONT_FT = 375;

    private KocsmaPanel kocsmaPanel;
    private List<Ital> italok = new ArrayList<>();

    public Vezerlo(KocsmaPanel kocsmaPanel) {
        this.kocsmaPanel = kocsmaPanel;
    }
```

```
    public void adatBeolvasas() {
        try {
            Ital.setValtoszamFontFt(ARFOLYAM_FONT_FT);
            AdatInput adatInput = new FajlbolInput(ITAL_ELERES);
            italok = adatInput.italLista();
            if (!italok.isEmpty()) {
                kocsmaPanel.itallapMegjelenites(Collections.
                                                    unmodifiableList(italok));
            } else {
                JOptionPane.showMessageDialog(kocsmaPanel, "Hibás adatfájl");
            }
        } catch (Exception ex) {
            Logger.getLogger(KocsmaPanel.class.getName()).log(Level.SEVERE,
                    null, ex);
        }
    }

    public void lokalitasEsBundleBeallitas(Locale lokalitas,
                                           ResourceBundle bundle) {
        Ital.setLokalitas(lokalitas);
        kocsmaPanel.lokalitasBeallitas(lokalitas);
        kocsmaPanel.szovegBeallitas(bundle);
        kocsmaPanel.repaint();
    }
}
```

A panel itallap-megjelenítő funkciója ugyanaz, mint a kocsmás feladat megoldásakor:

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

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

    private Locale lokalitas;

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

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

A „konstans” feliratokért a `szovegBeallitas()` metódus felel:

```
public void szovegBeallitas(ResourceBundle bundle) {
    lblItallapcim.setText(bundle.getString("ITALLAP_CIM"));
    rdbCsokkenoen.setText(bundle.getString("CSOKKENOEN"));
    rdbNovekvoen.setText(bundle.getString("NOVEKVOEN"));
}
```

Az események kezelésében semmi újdonság nincs:

```
private void rdbCsokkenoenActionPerformed(java.awt.event.ActionEvent evt) {
    csokkenoen();
}

private void rdbNovekvoenActionPerformed(java.awt.event.ActionEvent evt) {
    novekvoen();
}

private void lstItallapMousePressed(java.awt.event.MouseEvent evt) {
    valasztottItalKiirasa();
}
```

A rendezés gyakorlatilag ugyanaz, mint a kocsmás feladat esetében volt, hiszen a nyelvi különbségeket az `Ital` osztályba építettük be.

```
private void csokkenoen() {
    Ital.setValasztottSzempont(Ital.Szempont.FAJTA, Ital.CSOKKENOEN);
    italModel.sort();
    valasztottItalTorlese();
}

private void novekvoen() {
    Ital.setValasztottSzempont(Ital.Szempont.FAJTA, Ital.NOVEKVOEN);
    italModel.sort();
    valasztottItalTorlese();
}

private void valasztottItalTorlese() {
   lstItallap.clearSelection();
   lblAdottItalAdatok.setText("");
}
```

A választott ital adatainak kiíratásába viszont már beleszól a lokalitás. Mivel a nyelvváltoztatáskor óhatatlanul megváltozik a rendezési sorrend is, ezért a lokalitás megadásakor értelemszerűen ismét meg kell hívni valamelyik rendezést.

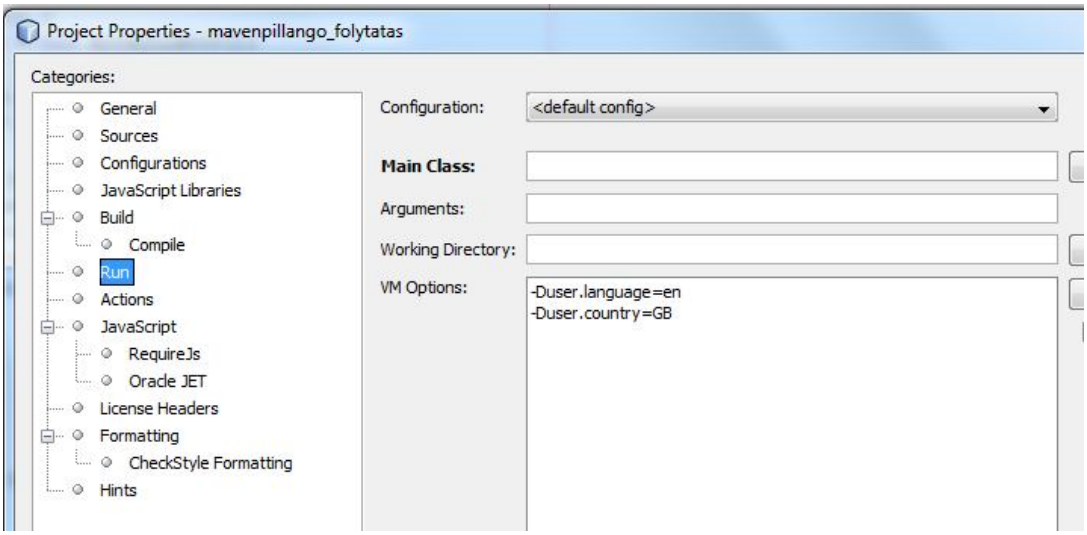
```
    private void valasztottItalKiirasa() {
        Ital valasztott = lstItallap.getSelectedValue();
        if(valasztott != null){
            NumberFormat numberFormat = NumberFormat.getInstance(lokalitas);
            NumberFormat penzFormatum = NumberFormat.
                                                getCurrencyInstance(lokalitas);
            lblAdottItalAdatok.setText(
                        numberFormat.format(valasztott.getDefaultDl()) + " dl "
                    + penzFormatum.format(valasztott.rendelesAr()));
        }
    }

    public void lokalitasBeallitas(Locale lokalitas) {
        this.lokalitas = lokalitas;
        if(rdbCsokkenoen.isSelected()) csokkenoen();
        if(rdbNovekvoen.isSelected()) novekvoen();
        valasztottItalKiirasa();
    }
}
```

Megjegyzés:

Ha induláskor már eleve az angol változatot szeretné, akkor pl. NetBeans-ben így lehet beállítani:

Projekt, jobb egérgomb, properties, és megkérjük a VM-t (virtual machine) hogy a megadott nyelven futtassa a programot:



ezt kell begépelni: -Duser.language=en -Duser.country=GB

Ehhez persze az kell, hogy a panel feliratait is a bundle fájlból vegyük. Ezeket kódból is beállíthatjuk, hasonlóan a frame címének beállításához, vagyis úgy, hogy a frame `start()`

metódusában meghívjuk a panel `szovegBeallitas()` metódusát, de beállíthatjuk a design módban is, így: