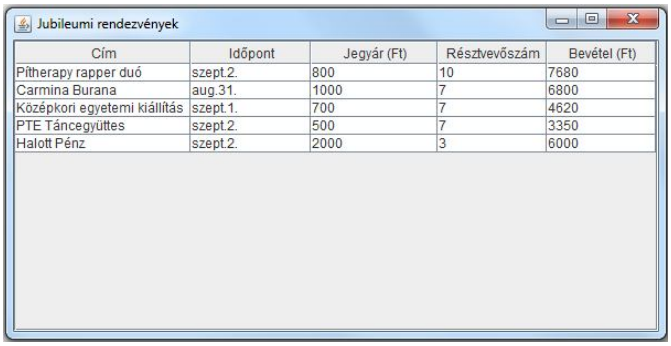
TUDNIVALÓK:
Most is és a következő gyakorlatokon is – akkor is, ha külön nem emeljük ki – az órán meg nem oldott feladatok HÁZI FELADAT-ként megoldandóak!!!
Ez fontos a tárgy sikeres teljesítéséhez!

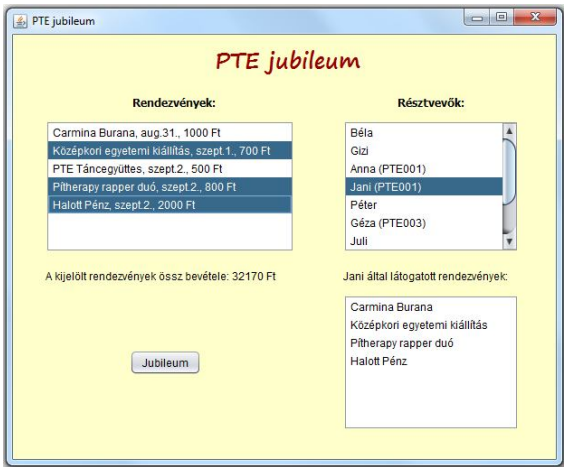
1. feladat:

a) „Fejezzük be” az előző órai feladatot, és beszéljük meg, hogyan lehet kiíratni a jubileumi rendezvényeket egy táblázatba:



Cím	Időpont	Jegyár (Ft)	Résztvevőszám	Bevétel (Ft)
Pítherapy rapper duó	szept.2.	800	10	7680
Carmina Burana	aug.31.	1000	7	6800
Középkori egyetemi kiállítás	szept.1.	700	7	4620
PTE Táncegyüttes	szept.2.	500	7	3350
Halott Pénz	szept.2.	2000	3	6000

b) Most alakítsuk át a múltkori megoldást grafikus felületű alkalmazássá, mégpedig úgy, hogy lehetőleg minél többet használjunk a meglévő megoldásból.



Az ábrán látható külsejű alkalmazás a következőket tudja:

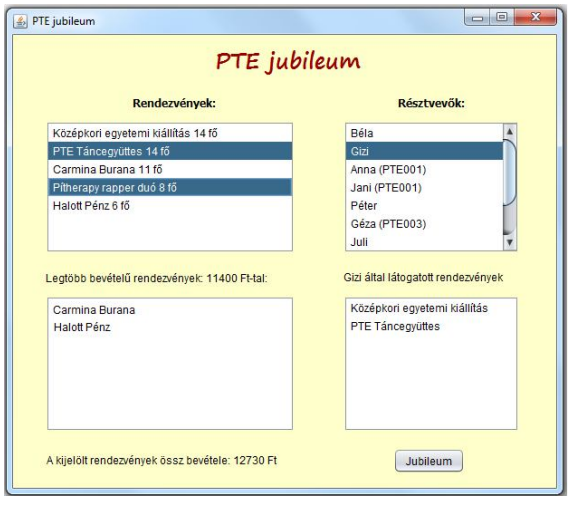
A program indulásakor már legyenek olvashatóak a rendezvények és a résztvevők listájának adatai, illetve adatbeolvasáskor minden résztvevő kapjon valamennyi pénzt.

A Jubileum feliratú gomb megnyomására fusson le az a szimuláció, amely már korábban is feladat volt, vagyis az, hogy minden rendezvény esetén mindegyik résztvevő „döntse el” véletlenül, hogy részt akar-e venni a rendezvényen vagy sem.

A rendezvények listája alatt lehessen látni a kijelölt rendezvények össz-bevételét.

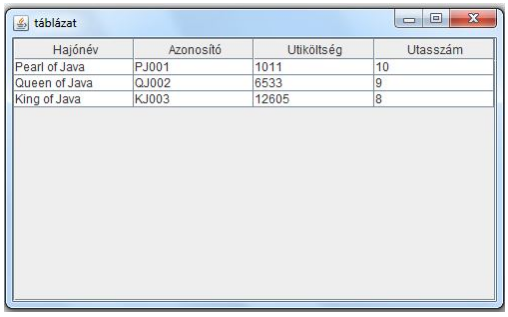
A résztvevők listája alatt pedig azt, hogy a kiválasztott résztvevő mely rendezvényeken vett részt. Mindkét felirat csak akkor jelenjen meg, ha már választottunk a megfelelő listából.

Bővítsük a feladatot: a Jubileum gomb hatására jelenjen meg a legtöbb bevételt hozó rendezvények listája is, maguk a rendezvények pedig legyenek résztvevőszám szerint csökkenően kiírva (most úgy, hogy a rendezvény neve és a létszám).

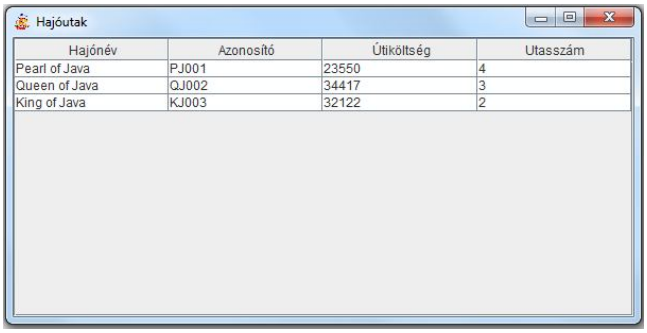


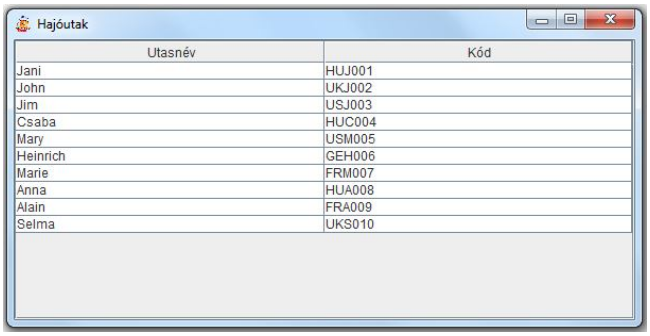
2. feladat:

Még mindig JAVA-TRAVEL. Próbáljuk meg ilyen formában is kiíratni az eredményt:



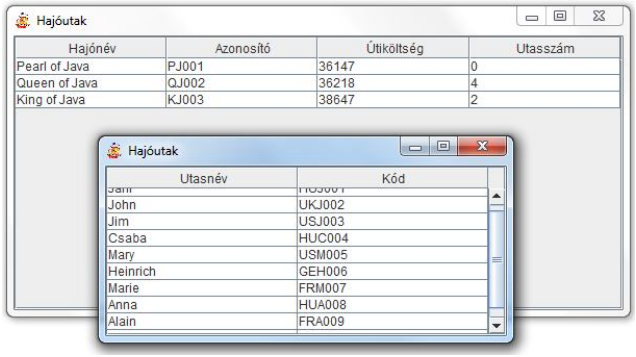
Önállóan bővítse tovább a feladatot, és most úgy oldja meg, hogy a hajók helyett az utasok adatait írja ki egy táblázatba.





Most próbálja meg úgy, hogy mindkét ablak megjelenjen

Most úgy, hogy az utaslistát tartalmazó ablak bezáró gombjára kattintva ne fejeződjön be a futás, csak tűnjön el az ablak, a hajóutak listáját tartalmazó ablakra kattintva viszont záródjon be az alkalmazás.

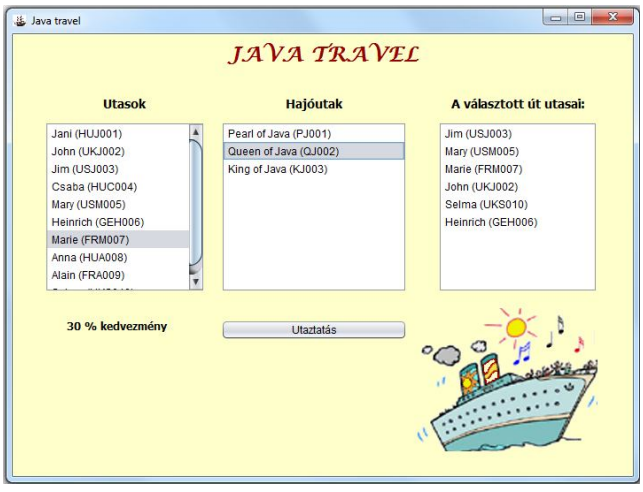


Ezeket még a konzolos alkalmazásból próbálja megoldani, mert segít a későbbiek megértésében, és az eddigiekhez képest nem nagy munka.

3. feladat

Alakítsuk át a feladatot grafikus felületűvé.

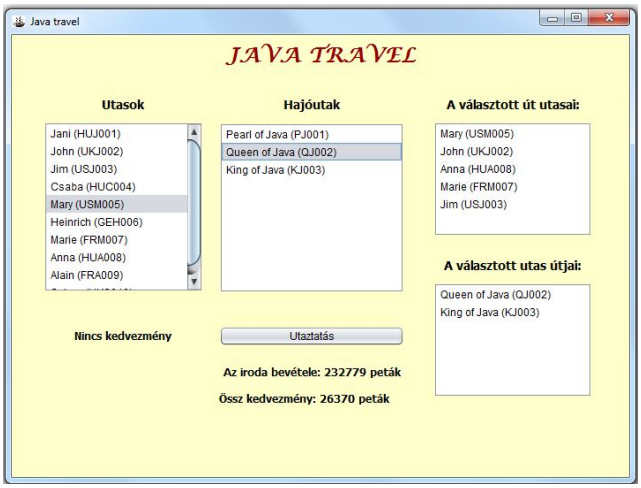
A felület 700*500-as méretű, induláskor látható a potenciális utasok és a hajóutak listája. Az utasok listájának bármelyik tagjára kattintva, a listafelület alatt olvasható, hogy az illető hány % kedvezményt kap, vagy az, hogy nincs kedvezmény.



Az Utaztatás feliratú gombra kattintva fusson le a korábbi projekt vezérlésének utaztatas() metódusa. Egy-egy hajóútra kattintva a harmadik listafelületen jelenjen meg az illető hajó utasainak listája.

A megoldás kapcsán strukturáljuk át a projektet: használjunk több csomagot, illetve vegyük külön az adatbeolvasást a vezérléstől. Az adatbevitelhez írjunk egy AdatInput nevű interfészt, és ezt implementálja majd a fájlból való olvasásra írt osztály. A fájlból való olvasást most úgy oldjuk meg, hogy az adatfájl része legyen a projektnek.
Önálló feladat: próbálja meg kódismétlés nélkül megoldani a fájlokból való olvasást.

Folytassuk a feladatot:



- Ha már költségekbe veri magát, akkor utazhasson az utas, vagyis az utazik() metódusa hatására az aktuális utat adja hozzá a hajóútjai listájához.

- A kedvezményes utas esetén azt is határozzuk meg, hogy útjai során összesen mennyi (hány peták) kedvezményt kapott a cégtől?

- Tesztelje a kedvezmény kiszámítási módjára írt kódot.

- Ha jól működik a metódus, akkor az Utaztatás feliratú gomb hatására a gomb alatt jelenjen meg a cég teljes bevétele és a kedvezményekre fordított összeg is.

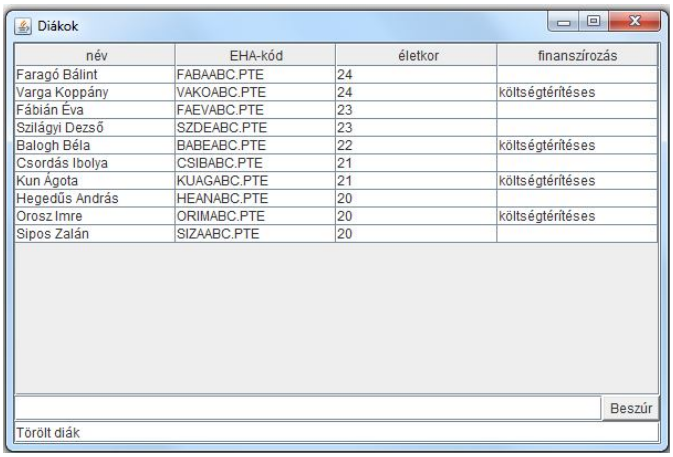
FONTOS MEGJEGYZÉS: Meglehetősen pazarló (redundáns) megoldás az, hogy minden hajóút objektum tartalmazza a hozzá tartozó utasok listáját, és minden utas objektum a hozzá tartozó utak listáját, azaz sok adatot duplán is tárolunk.
Feladat: gondolkozzon el rajta, és próbálja megoldani, hogy nem redundáns módon lehessen tárolni az adatokat. (Az adatbázisból tanultak segítenek, de elég a józan paraszti ész is ☺)

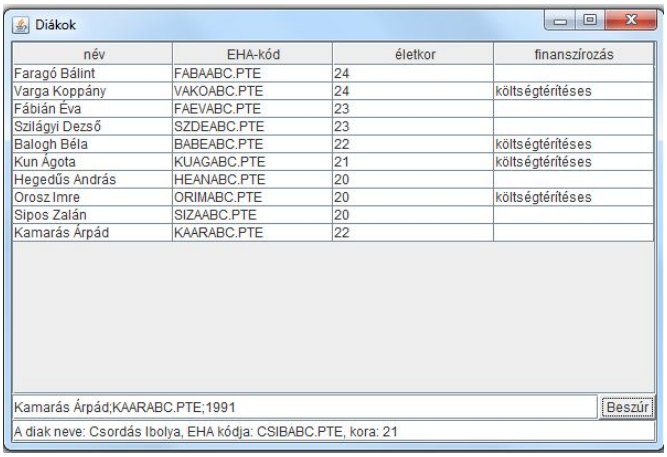
4. feladat – szorgalmi:
Nem egyszer lehet szükség rá, hogy konzolos alkalmazás kimeneteként hozzunk létre pl. egy táblázatot, ezért van értelme az ilyen, nem „generálós” feladatnak.

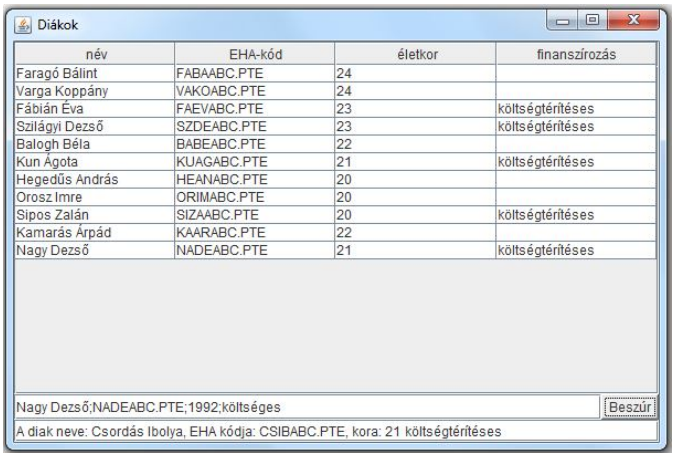
Vannak diákok és költségtérítéses diákok. Mindegyikőjük esetén meg kell adnunk a nevét, nepgun-kódját, születési évét. A diákok adatait a *diakok.txt* fájl tartalmazza. Olvassa be őket, és minden egyes diák esetén véletlenszerűen döntse el, hogy az illető költségtérítéses-e vagy sem. Vegye figyelembe, hogy a diákok kb. 40%-a lehet költségtérítéses.

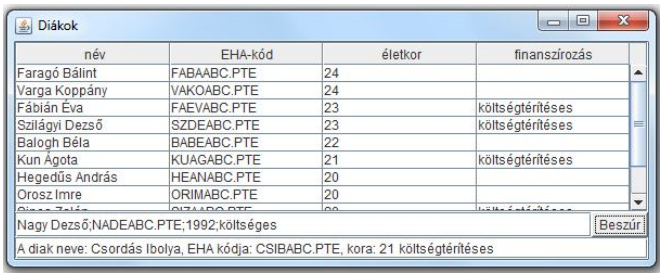
Írassa ki az adatokat egy ablakba, majd az ablak alján hozzon létre egy beviteli mezőt, mellette egy „Beszúr” feliratú gombbal, alatta pedig egy másik, nem szerkeszthető szövegmezőt.
A gombnyomás hatására kerüljön be a táblázatba a gomb előtt lévő szövegmezőbe írt adatok alapján készült sor. (Az egyszerűség kedvéért most feltételezhetjük, hogy helyesen töltik ki a sort *név;kod;szülév* vagy *név;kod;szülév;valami* formában – ha van negyedik adat is, akkor

költségtérítéses az illető. Esetleg dobhat hibaüzenetet, ha nem ilyen a beírt sor. (Hibaüzenet: `JOptionPane.showMessageDialog(this, "üzenet");`)
Ha rákattintunk a tábla valamelyik sorára, akkor a sor törlődjön ki, és a kitörölt diák adatai jelenjenek meg az alsó szövegmezőben.









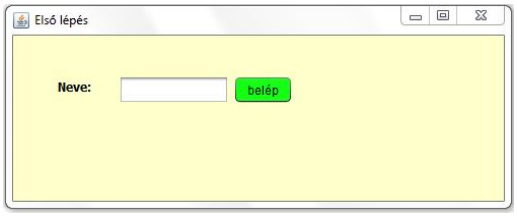
Természetesen scrollozható.

Segítség:
A komponenseket előbb rakja egy (vagy több) panelre – attól függően, hogy hányra teszi, a panel vagy FlowLayout() vagy BorderLayout() elrendezésű legyen.
A gomb figyelésére ActionListener()-t lehet rendelni, ennek ActionPerformed() metódusa kezeli az eseményt, a táblához egy MouseListener()-t lehet rendelni, itt a MouseClicked() metódust lehet használni.
Törölni, beszúrni a táblamodellből/be lehet.

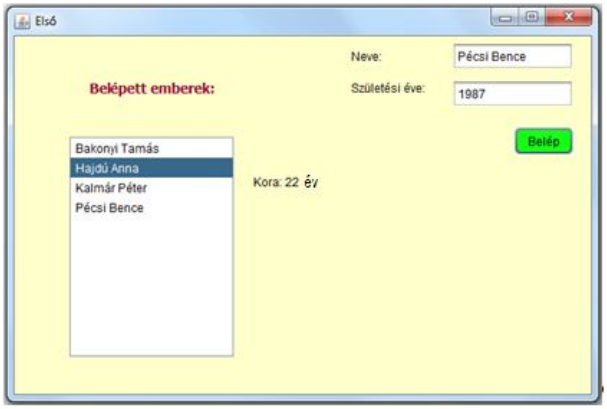
További (egyszerű) feladatok:

5. feladat:
Első lépésként hozzunk létre egy „beléptető” rendszert, vagyis az 500×200-as felületen a nyomógomb megnyomásának hatására üdvözöljük a szövegmezőbe írt nevű embert. Üres mező esetén adjon hibaüzenetet. (Csak akkor, ha nem unja ☺)

6. feladat:
Játsszunk kicsit, és módosítsa az előző feladatot úgy, hogy ha az egérrel a nyomógomb fölé megyünk, akkor annak háttérszíne változzon meg zöldre, ha viszont az egér elhagyja a nyomógomb területét, akkor a színt változtassuk vissza.



7. feladat:
Az előzőek folytatásaként most egy listában jelenítsük meg a beléptetett emberek nevét.
Belépni a „Belép” feliratú gomb hatására lehet, ekkor a megadott nevű, születési évű ember bekerül a listába. Hibás születési év esetén adjon hibajelzést. (Nem szám, negatív, nagyobb, mint az aktuális dátum (évszám).)
Ha sikerült, akkor a listára kattintva, a lista mellett jelenjen meg a kiválasztott ember életkora. (Most egy fix helyre kerüljön, de azt is megoldhatja, hogy mindig a kiválasztott ember neve mellett jelenjen meg a kora.)

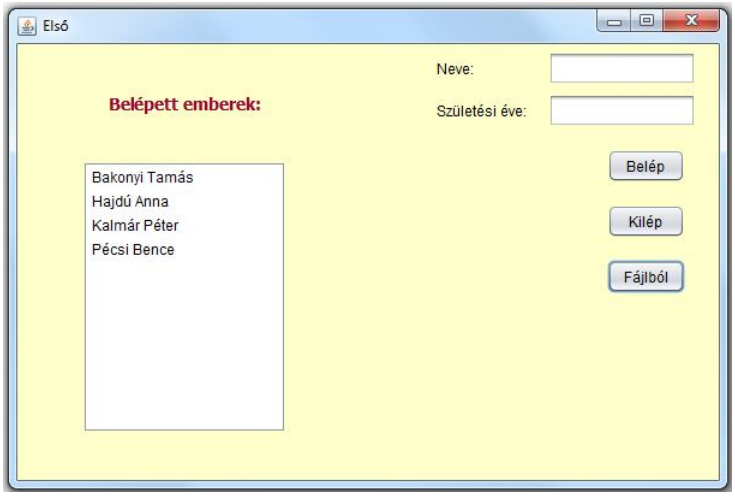


Figyeljen rá, hogy mindenki csak egyszer kerülhessen be a listába.
Sikeres adatbevitel után ürítse ki a beviteli mezőket, és fókuszáljon a névmezőre.

8. feladat
a/ A „Kilép” gombra kattintva törölje ki a listából a kijelölt embert. Ekkor a korára vonatkozó felirat is tűnjön el.

b/ Oldja meg, hogy egyszerre több embert is törölhessen a listából.

c/ A „Fájlból” gomb hatására fájlból töltse be az adatokat.
A fájl egyelőre lehet az src mappa fix helyén, de ha kedve és ideje van, akkor utánanézhet, hogy hogyan lehet interaktív módon kiválasztani a fájlt.

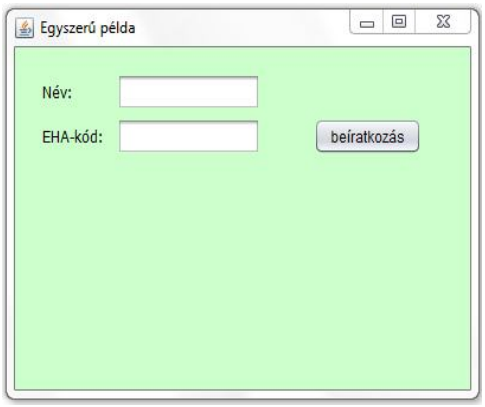


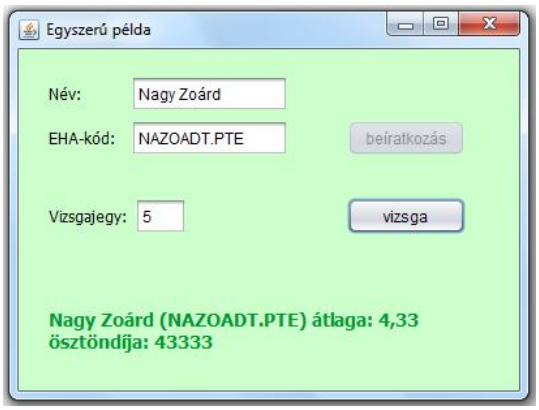
9. feladat

A program egyetlen diákot tud kezelni. Ha megadjuk a nevét, neptun-kódját, majd rákattintunk a beiratkozás gombra, akkor jelenjen meg a többi komponens is, és vizsgázhasson a diák (a beiratkozás gomb viszont váljon inaktívvá) . A legelső vizsga után jelenjenek meg az adatai is, vagyis a neve, neptun-kódja (ne zavarja, hogy a képen EHA-kód szerepel ☺), átlaga, ösztöndíja (illetve adott esetben az, hogy nem kap ösztöndíjat). Természetesen újabb vizsga esetén módosulnak az adatok.

A szükséges statikus változókat konstansként adjuk meg.

A diák akkor kap ösztöndíjat, ha átlaga egy egységes határ fölött van, ekkor az ösztöndíj az átlag valahányszorosa – természetesen a szorzó is egységes érték.



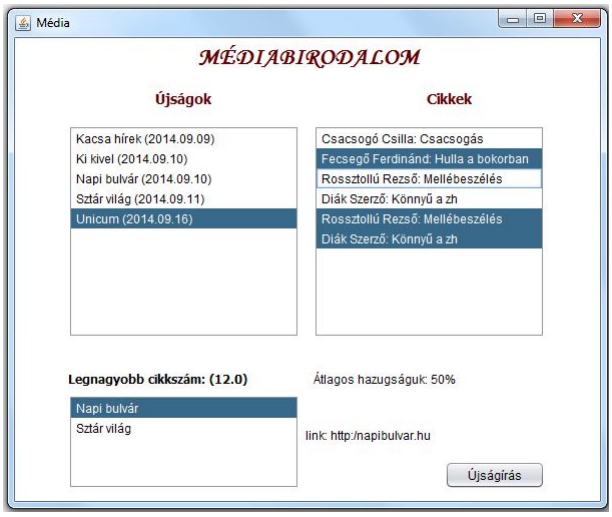


Kicsit összetettebb feladatok:

10. feladat

Ennek a korábbi feladatnak a „grafikásítása” lesz a cél.
Az osztály-leírásokat, a hivatkozott ujsagiras() metódus leírását megtalálja a korábbi feladatsorokban (gyak_2 feladatsor, 2.a, 2.b feladat).
(De van egy rövid emlékeztető a következő feladatban is.)

Az adatokat természetesen fájlból olvassuk.



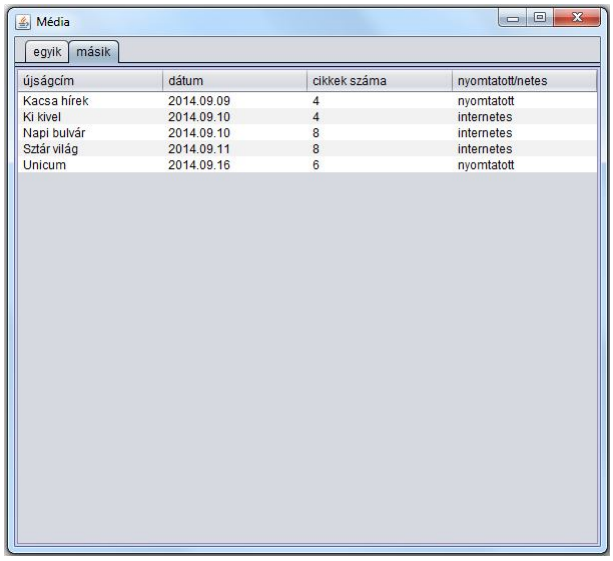
a/ Írassuk ki az újságokat névsorba rendezve.
(Az ábrán látható frame mérete: 600 × 500.)

Az újságírás gomb hatására hívjuk meg a korábban megírt `ujsagiras()` metódust, amelynek hatására a beolvasott cikkek közül véletlenszerűen bekerül néhány egy-egy újságba.

Az újságra kattintva a másik oldalon lássuk a benne megjelent cikkeket. Egyszerre csak egy újságot lehessen kiválasztani.
A cikkek közül viszont akár többet is, a lista alatt legyen látható a kiválasztott cikkek átlaghazugság értéke.

Az újságokat tartalmazó lista alatt azoknak az újságoknak a nevét lehessen olvasni, amelyekben a legtöbb cikk jelent meg. Ebből a listából is csak egyetlen nevet lehessen választani, és ha ez internetes, akkor a lista mellett jelenjen meg az újság linkje, ha nyomtatott, akkor a példányszáma.

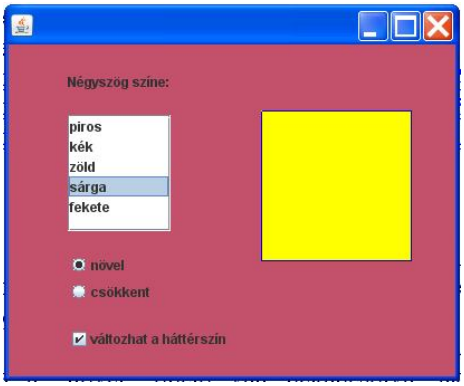
b/ Oldja meg, hogy regiszteres módon tudjuk kezelni az újságokat. Az első fülre kattintva az előző felületet lehessen látni, a másodikra kattintva a gyak_3 feladatsorban szereplő változatot, vagyis egy táblázatot.



Segítség: a tab-fülek miatt a frame-t kicsit nagyobb méretűre kell megadni, kb. 550-es magasságúra.
A táblázatot rárakhatja az előző órán vett Ablak osztály pici módosításával, de lehet úgy is, hogy a palettán található Table elemet használja – járjon utána, hogy ezt hogyan lehet, de majd a kiadott segédletben erről is lesz pár szó.

Majdnem végül: két kis játék:

11. feladat
Hozza létre a mellékelt felületet (a négyzet egy bekeretezett (border) JLabel).
A négyzet felülete változzon a listából kiválasztott színűre.
Ha a „növel” opció van bekapcsolva, akkor a négyzet belsejére kattintgatva növekedjen annak mérete, ha a csökkent opció van bekapcsolva, akkor pedig csökkenjen.
Ha be van kapcsolva a „változhat a háttérszín” választás, akkor a háttérre kattintva annak véletlenszerűen változzon meg a színe, ha nincs bekapcsolva, akkor ne történjen semmi.



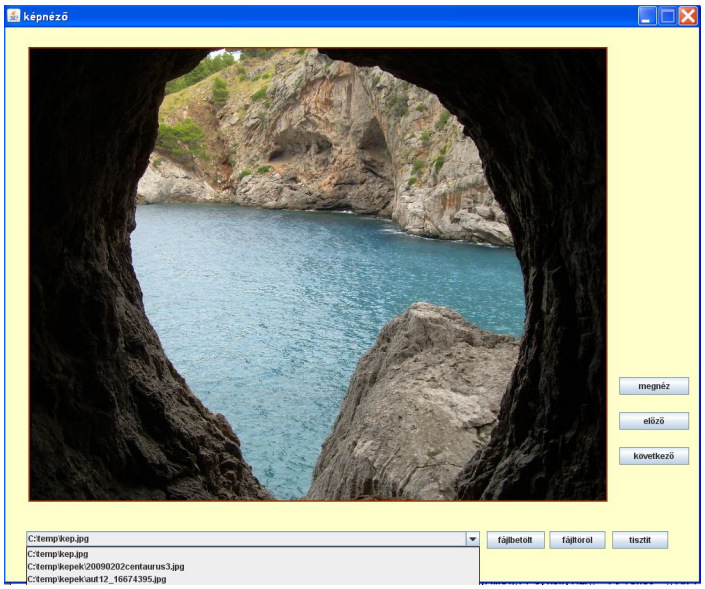
Emlékeztető:
A JRadioButton-okat Button Group-ba kell rakni, hogy egyszerre csak az egyik választás éljen.

Átlátszóság: ha az opaque tulajdonság ki van kapcsolva, akkor a komponens átlátszó.

12. feladat

Hozzon létre egy képnézegetőt!
A nem változtatható méretű lap központi részét egy keretezett (border) JLabel tölti ki, ide kerülnek majd a képek. Előtte azonban töltsük fel a lap alján látható combo-boxot képfájl-címekkel. Természetesen jelezzük az esetleges hibákat.
Az induló könyvtár legyen az aktuális.
Használjon file-filtert, és csak képfájlokat engedjen kiválasztani. (Lehet csak jpg vagy jpeg.)

(javax.swing.filechooser csomag, FileFilter osztály; a Help-ből el lehet jutni egy jól használható mintapéldához.)

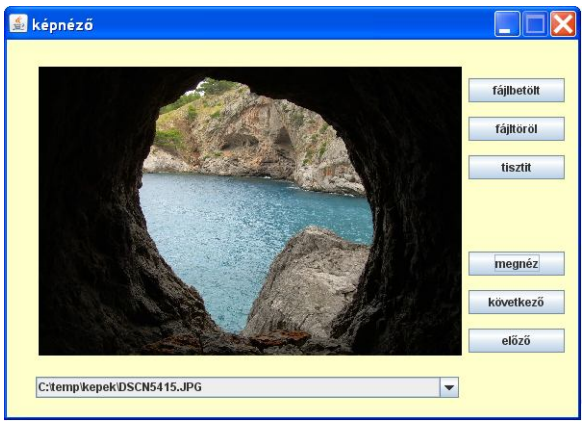


A betöltött fájlokból természetesen törölhetünk – egyenként is, illetve a „tisztít” gomb hatására egyszerre az összeset is.

A kiválasztott fájlt a „megnéz” gomb hatására nézhetjük meg.
Az „előző” gomb hatására az előző képeket láthatjuk, a „következő” hatására a következőket.
Legyen mindkét gomb „végtelenített”, vagyis ha pl. az előrehaladással eljutottunk a legelső képhez, akkor az utolsóval folytassa, és fordítva, az utolsó után következzen az első.

Segítség: Használja a JLabel setIcon(ImageIcon) metódusát – természetesen helyes szintaktikával. (Ez nem az, csak javaslat.)

Előny: könnyen alkalmazható.
Hátrány: fix képméret, vagyis a nagyobb képekből csak részlet látszik. (Ezért is ilyen nagyméretű a lap.)
De aki megtalálja a megoldást (az első ilyen ember ☺) kap egy tábla csokit.
Vagyis: ne a label méretét alakítsuk a képhez (ez viszonylag egyszerű), hanem a kép méretét a labelhez.



Módosítsa úgy a megoldást, hogy jóval kisebb felületen is látható legyen a teljes kép.
Segítség: Használja a grafikus felület drawImage(kep, x, y, szelesseg, magassag, null) metódusát.

Elvileg elkérheti a label grafikus felületét, és arra rajzolhat, de ez nem igazán szerencsés megoldás, mert nem stabil – egy másik alkalmazás könnyedén le tudja törölni.
Helyesebb megoldás, ha a komponens paintComponent() metódusát használja. (De gondolja végig, hogyan lehet módosítani egy meglévő osztály metódusát.)

Ha ez működik, akkor még arra is figyeljen, hogy ne torzuljanak a képek, vagyis őrizzék meg eredeti arányukat, de férjenek bele a megadott keretbe.

Megjegyzés: Képeket nem csak labelre lehet tenni, lesz majd szó ügyesebb megoldásról is.

Általános segítség:

1. Az újrahasznosíthatóság elve nagyon fontos az OOP szemléletben. Ezért sokkal elegánsabb, ha a panelt nem „drótozzuk be" a vezérlő osztályba, hanem külön kezeljük. Ezért hozzunk létre egy feluletek csomagot, és ebben egy JPanel form-ot. Erre húzzuk fel a megfelelő komponenseket, és értelmezzük a szükséges eseményeket.
De persze, ennek a panelnek valahogy rá kell kerülnie a frame-re. Ezért előbb fordítsuk le a projektet. Ha sikerült, akkor ugyanúgy rá lehet húzni a saját panelt a frame-re, mintha az „gyári panel" lenne. Ennek feltétele az, hogy a panelnek legyen paraméter nélküli konstruktora, vagyis úgynevezett JavaBean legyen, és persze, le is forduljon.

2. Listakezelés:
Az úgynevezett modell-view-controller (MVC) szemlélet értelmében különválasztjuk az adat-modellt, a felületet és a vezérlést. Ez a helyzet a listakezeléssel is. Az adatmodell esetünkben egy – mondjuk – italokból álló speciális lista lesz, az ú.n. `DefaultListModel`.
Deklarációja:
```
private DefaultListModel<Ital> modell = new DefaultListModel<>();
```
(vagyis csaknem ugyanolyan, mint bármelyik más listadefiníció)

Ezt a modellt hozzá kell rendelnünk a listafelülethez (`JList`). (Vigyázat, kétféle listáról van szó: a `JList` egy grafikus komponens, a swing része, a `List` pedig a util csomag listakezelésre alkalmas interfésze).
A hozzárendelést célszerű a konstruktorban megoldani, ha már létrejött a modell, egyébként pedig akkor, ha létrejön:
```
sajatJLista.setModel(modell);
```

A szétválasztás lényege: adatot mindig a modellbe rakunk, innen törlünk, itt manipuláljuk, megjelenni azonban a `JList`-ben jelenik meg (mégpedig az adott objektum `toString()`-je), itt tudjuk kiválasztani a manipulálandó elemet, stb.

Természetesen ugyanez a helyzet, ha más típusú adatokat akarunk kezelni. ☺