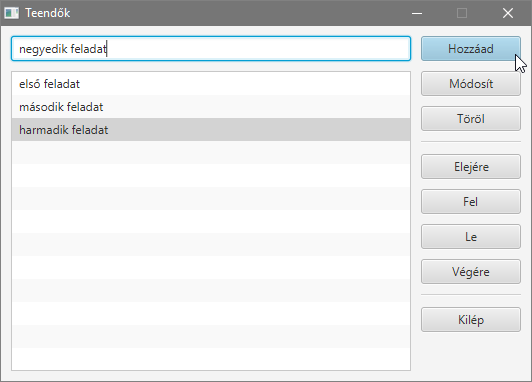
# Teendők

## Feladat

Ebben a leckében egy to-do list programot készítünk, amelyben teendőinket, feladatainkat tarthatjuk nyilván.



A felső szövegmezőbe írhatjuk a következő feladatot, és a Hozzáad gombbal adhatjuk a lista végéhez.

Ha a listában kijelölünk egy elemet, a tartalma beíródik a szövegmezőbe. Ha megváltoztatjuk, majd megnyomjuk a Módosít gombot, a kijelölt elem helyett beíródik a módosított.

Amikor elkészültünk egy feladattal, kijelöljük, majd a Töröl gombbal töröljük a listáról.

A feladatok sorrendjét módosíthatjuk az Elejére, a Fel, a Le és a Végére gombokkal.

Kilépéskor a program elmenti a lista tartalmát egy *teendok.txt* nevű fájlba.

Induláskor megpróbálja betölteni a legutóbb mentett listát. Ha ez nem sikerül, akkor üres listával indulunk.

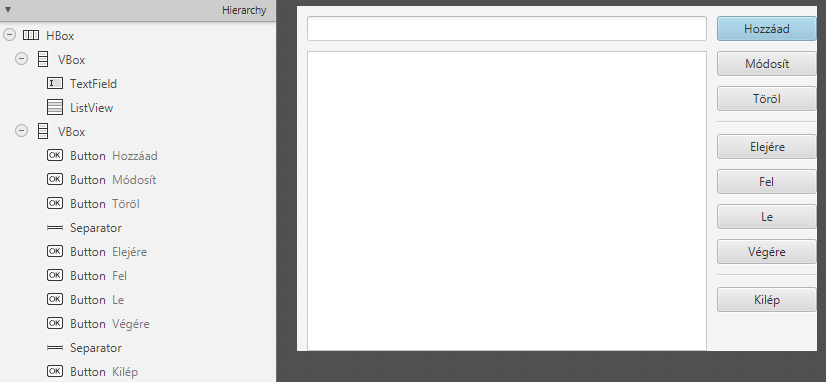
## Ablak elkészítése

Hogyan rendeznéd el az elemeket az ablakban?

Kezdj egy új JavaFX FXML projektet *teendok* néven!

Két egymás melletti VBox -szal fogjuk megoldani az elhelyezést. A listához a ListView vezérlőt használjuk.

Készítsd el a program ablakát a Scene Builderben az alábbi minta alapján:



A HBox preferált szélessége és magassága számított, térköze 10 pont legyen! Bal oldalon és felül legyen 10 pontos padding-je! A másik két oldalon maradjon 0 a padding! Ez így nem szép, de ha majd letiltjuk az ablak átméretezését, akkor automatikusan ide is kerül két 10 pontos padding.

Mindkét VBox térköze 10 pontos legyen, szélessége és magassága számított. Ezeknél már nem kell padding és margin. A jobb oldali VBox preferált szélessége 100 képpont legyen!

A TextField preferált szélessége legyen 400 pont, a ListView pedig legyen 400x300-as! A szövegmező azonosítója legyen txtUjFeladat, a listáé pedig lstFeladatok!

A gombok és az elválasztók preferált szélessége legyen 100 képpont!

A Hozzáad gomb legyen Default Button, a Kilép gomb pedig Cancel Button!

Az egyes gombok megnyomása a következő metódusokat futtassa: hozzaad, modosit, torol, elejere, fel, le, vegere, kilep!

Add meg a controller osztály nevét a bal alsó sarokban (Document, Controller)!

Ezután nyisd meg a View, Show Sample Controller Skeleton paranccsal a controllerbe másolandó kódot, és másold a vágólapra a szükséges deklarációkat!

Mentsd a jelenetet, és válts vissza a NetBeans-re!

Állítsd be az ablak címét (Teendők), és tiltsd le az átméretezését!

Nézd át a generált FXML dokumentumot!

## A controller osztály

Másold be a kódot a vágólapról a controller osztályba! Töröld az eseménykezelők ActionEvent paramétereit! Végezd el a szükséges importálásokat!

A ListView-nál meg kell adni a listaelemek típusát (általában String), ugyanúgy, mint az ArrayList-nél:  


Futtasd a programot!

## A ListView használata

A ListView-hoz automatikusan létrejön egy olyan lista, amely az elemeket tárolja (típusa ObservableList). Ez hasonlít az ArrayList-hez, de elemei automatikusan megjelennek a ListView-ban. A ComboBox is egy ilyen listával működik.

A listát a vezérlő getItems metódusával kaphatjuk meg.

Az ArrayList metódusai használhatók az ObservableList esetén is: add, addAll, get, set, remove, clear, contains, isEmpty, size.

Melyik mire való?

Nézzünk néhány példát:

@FXML  
private ListView<String> lista;   
// több elem hozzáadása  
lista.**getItems().addAll**("sonka","szalámi","sajt");  
// egy elem hozzáadása  
lista.**getItems().add**("gomba);  
// 0-dik elem tartalma  
lista.**getItems().get**(0);  
// 0-dik elem módosítása  
lista.**getItems().set**(0,"hús”);  
// 0-dik elem törlése  
lista.**getItems().remove**(0);

A listához tartozik egy kijelölésmodell objektum is. Ennek metódusaival lehet lekérdezni vagy módosítani a kijelölt elemet vagy elemeket. Alapértelmezésben egy elem jelölhető ki, de ez módosítható.

A kijelöléssel kapcsolatos műveletekhez le kell kérni a kijelölésmodellt a getSelectionModel metódussal.

Nézzünk erre is néhány példát:

// 0-dik elem kijelölése   
lista.**getSelectionModel().select(0)**;   
// kijelölt elem indexe (ha nincs kijelölve: -1):  
int i = lista.**getSelectionModel().getSelectedIndex()**;   
// kijelölt elem tartalma (ha nincs kijelölve: null):  
String s = lista.**getSelectionModel().getSelectedItem()**;

A kijelölés változását a selectedItem property eseménykezelőjének segítségével figyelhetjük.

Az eseménykezelőt, amely a kijelölés változásakor fut le, így adhatjuk meg:

lista.**getSelectionModel().selectedItemProperty().addListener**(  
 (v, regi, uj) -> {  
 // *utasítások*

});

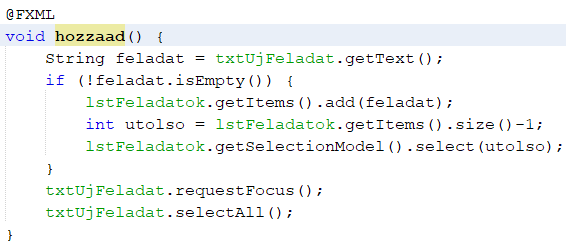
## Feladat hozzáadása, módosítása, törlése

A Hozzáad gomb megnyomásakor a program kiolvassa a szövegmezőbe írt szöveget, és ha ez nem üres, akkor hozzáadja a listához.

Mivel az új elem a lista végére kerül, kijelöli az utolsó elemet.

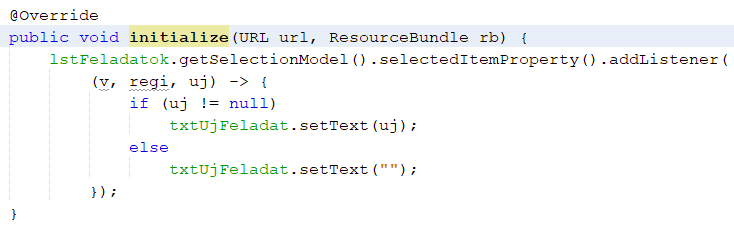
Végül a kurzort a szövegmezőbe viszi a requestFocus metódussal, és kijelöli a szöveget, hogy egyből lehessen írni a következő feladatot.

Készítsd el a hozzaad metódust:



Próbáld ki!

Azt szeretnénk, hogy a listában kijelölt elem megjelenjen a felső szövegdobozban. Ehhez egy eseménykezelőt kell készíteni az initialize metódusban, amely a kijelölt elem változásakor bemásolja az új elemet a szövegdobozba:



Az uj paraméter értéke akkor lesz null, ha letöröljük az utolsó elemet is a listából. Ebben az esetben üresre állítjuk a szövegmezőt.

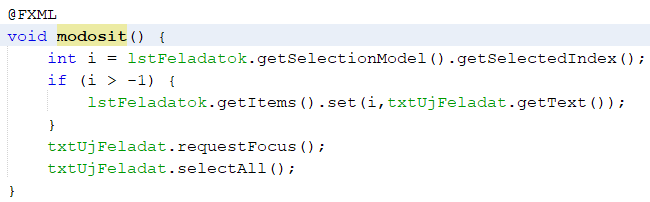
Próbáld ki! Milyen módszerekkel (billentyűkkel, egérrel) lehet egy elemet kijelölni a listában?

A kijelölt feladat módosításához először meghatározzuk a listában kijelölt elem indexét. Ha nincs semmi kijelölve, akkor ez -1, egyébként ennél nagyobb.

Ha ki van jelölve elem, akkor elvégezzük a módosítást.

Utána a kurzort visszavisszük a szövegmezőre, és kijelöljük a szöveget.

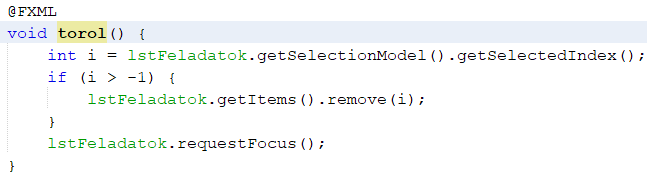
Készítsd el a modosit metódust:



Próbáld ki!

A törlést is akkor tudjuk elvégezni, ha ki van jelölve egy elem. A törlés után a listára visszük a kurzort.

Készítsd el a torol metódust:



Próbáld ki!

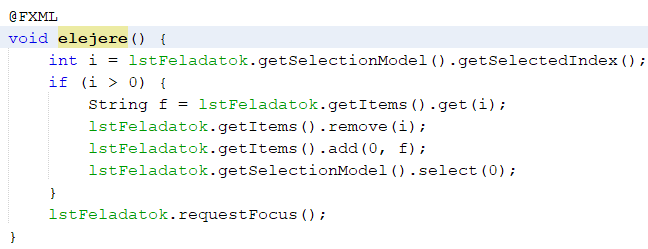
## Listaelemek áthelyezése

Hogyan lehet a listában áthelyezni egy elemet?

Először a lista elejére mozgatást nézzük meg! Ez akkor lehetséges, ha a kijelölt elem indexe 0-nál nagyobb.

Az áthelyezést úgy végezzük, hogy a lista i-dik helyéről töröljük az elemet, majd beszúrjuk a 0-dik helyre, és ki is jelöljük.

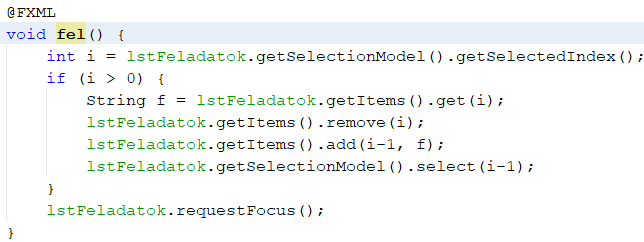
Készítsd el az elejere metódust:



Próbáld ki!

A felfelé mozgatás hasonlóan történik, csak most az i-1-dik helyre szúrjuk be az i-dik helyről törölt elemet.

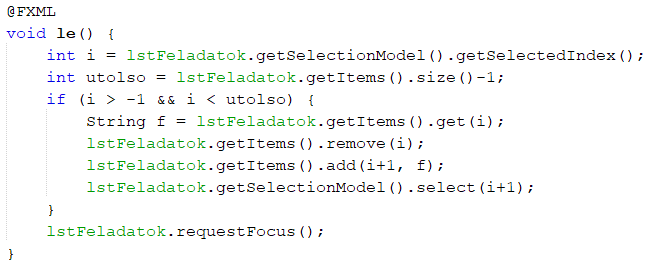
Készítsd el a fel metódust:



Próbáld ki!

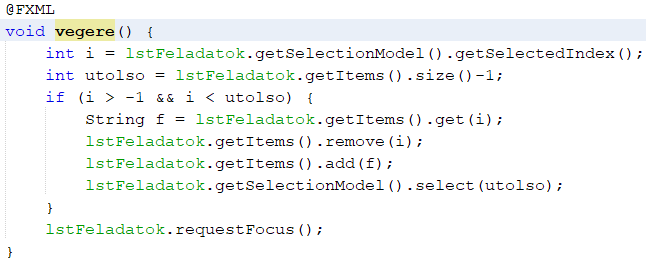
Lefelé mozgatásnál az i-dik helyről törölt feladatot az i+1-dikre szúrjuk be. Ez azonban csak akkor lehetséges, ha nem az utolsó elem volt kijelölve.

Készítsd el a le metódust:



Próbáld ki!

A kijelölt feladat lista végére helyezése is hasonlóan végezhető el:



Próbáld ki!

## Párbeszédpanelek

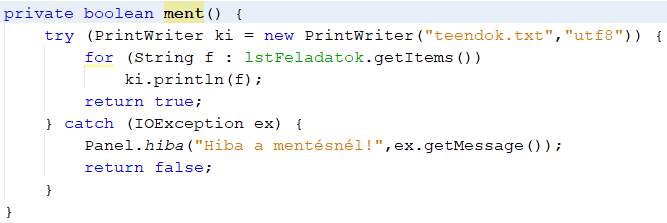
Készíts egy új, panel nevű csomagot a projektbe (jobb kattintás a Source Packages-re, New, Java Package)!

Másold az új csomagba a kapott Panel.java fájlt! Ez a korábbi leckékben is használt párbeszédpaneleket tartalmazza.

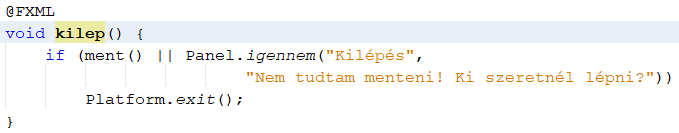
## Mentés és betöltés

A Kilépés gomb megnyomásakor a kilépés előtt szeretnénk megőrizni a lista tartalmát egy *teendok.txt* nevű, UTF-8 kódolású szöveges fájlban.

Készítsd el a ment metódust, amely a listában szereplő feladatokat a szöveges fájlba írja! Ha nem sikerül a mentés, hamis értéket adjon vissza, egyébként igazat!



A kilep metódusból meghívjuk a ment metódust. Ha sikerül a mentés, akkor kilépünk, egyébként megkérdezzük, hogy ki szeretne-e lépni:

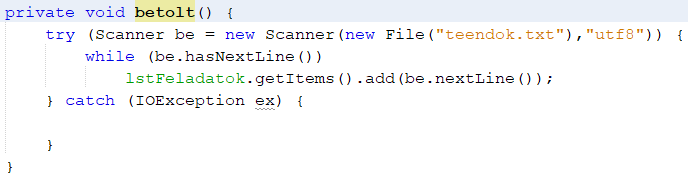


Próbáld ki! Keresd meg és ellenőrizd a *teendok.txt* fájlt!

Tedd írásvédetté a *teendok.txt* fájlt, és próbálj meg így is menteni! Utána kapcsold ki az írásvédettséget!

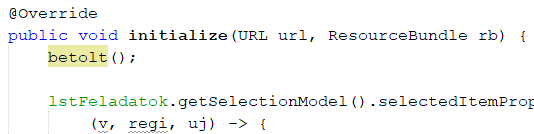
A program indításakor be kell tölteni a *teendok.txt* fájlból az adatokat. Ha nem tudja megnyitni a fájlt, akkor üres listával induljon a program!

Készítsd először a betolt metódust:



Figyeld meg, hogy a metódus hiba esetén nem csinál semmit!

Ezután hívd meg a betolt metódust az initialize metódusból:



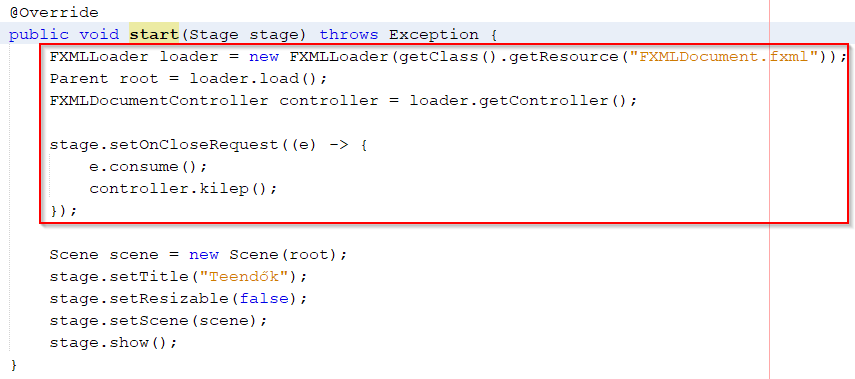
Próbáld ki! Írj be egy új feladatot, nyomd meg a Kilépés gombot, majd indítsd el újra!

## Kilépés

Az ablak bezárásakor is meg kell hívni a kilep metódust.

Mivel az eseménykezelőt a Teendok osztályban hozzuk létre, és a kilep metódus a controller osztályban van, szükségünk lesz egy változóra, amely a controller osztályra mutat.

Ehhez sajnos a Teendok osztályban át kell írnunk a start metódus elejét, és utána tudjuk elkészíteni az eseménykezelőt:



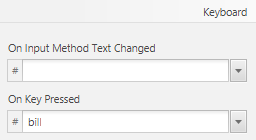
Az esemény consume metódusa megakadályozza az esemény továbbadását.

Próbáld ki! Zárd be az X-szel az ablakot, és ellenőrizd a mentést!

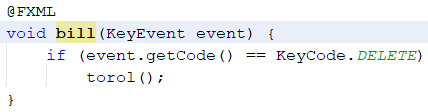
## Billentyűk kezelése

Azt szeretnénk, hogy a listában a kijelölt elemet a Delete billentyűvel is le lehessen törölni.

Nyisd meg a jelenetet a Scene Builder-ben, jelöld ki a ListView elemet, majd a Code lapon add meg, hogy egy billentyű lenyomásakor fusson le a bill metódus:



Ezután készítsd el a bill metódust a controller osztályban:



A KeyEvent esemény tartalmazza a megnyomott billentyű kódját, amelyet a getCode metódussal tudunk kiolvasni. A kódok állandóként érhetők el a KeyCode osztályban. (Beíráskor a pont után lehet választani.)

Próbáld ki!

Fontos! A Delete billentyű eseménykezelőjét csak akkor hívja meg a program, ha a listában állva nyomjuk meg a Delete billentyűt. Ha másik elemen állva is szeretnénk használni, akkor ahhoz az elemhez is hozzá kell rendelni a megfelelő eseménykezelőt.

## Kérdések

1. Hogyan tudjuk kezelni egy ListView elemeit a programban?
2. Hogyan lehet figyelni egy ListView-ban a kijelölés változását?
3. Hogyan lehet figyelni a programból egy billentyű megnyomását?
4. Hogyan lehetne továbbfejleszteni a programot?