Pásztornyilvántartó rendszer dokumentáció

Programozás alapjai 3.házi feladatához, Pete Dávid (GVJ529)

A JSON kezelésre egy külső könyvtárat használok, itt lehet letölteni: https://github.com/stleary/JSON-java

Módosulások az eredeti feltöltésem óta:

- Átírtam a programban az összes szöveget angolra, hogy ne legyenek karakterkódolási problémák
- A forráskódot sok helyen kommenteztem, ahol eddig nem volt elég, vagy csak nagyon hiányos volt
- Rajzoltam egy jóval részletesebb Osztálydiagram -ot
- Ide hozzáírtam a Fontos metódusok működésének leírása részt
- Bővítettem a Tesztek leírása részt
- Bővítettem az Adatszerkezetek rész leírását
- *(kisebb hibajavítás: ha törlök egy céget, akkor az eltűnik az összes pásztortól, akinek az a munkaadója)*

Osztályok, adatszerkezetek leírása

Fontos absztrakt osztályok

Próbáltam úgy elkészíteni az osztályaimat, hogy minél inkább kihasználhassam az öröklés és a heterogén kollekciók előnyeit, mivel sok, egymással szinte azonos működésű része van a programomnak. Ezek a részek a táblázatos nézetek, ahol egy entitást lehet hozzáadni módosítani és törölni (CRUD műveletek)

Ehhez valamilyen Model-View-Controller struktúra féleséget próbáltam létrehozni, ahol a **Controller** elsősorban az *ActionListner*t és vele a törlési és hozzáadási műveleteket végzi, a **Model** elsősorban a *JTable TabelMode*l-jének a leszarmazottja, és a **View** pedig egy olyan osztály, ami ha az aodott nézetre navigálunk akkor az azt felépíti, és hozzácsatolja a Modelt és Controllert.

abstracts/AbstractCRUDTableController.java

A controller ősosztályom, ami tehát elsősorban *ActionListner* a hozzáadás és a módosítás gombokhoz, és azokat ITT végre is hajtja.

Attribútumok:

- *TableViewBuilder view*
  - *Kontstruktorparaméter, Innen éri el a mindent: a JTable táblázatot, a TableModel-t és a ViewBuildert is*
  - *(a többi attribútum, ami a kódban ebből csak "ki let hibatkozva" az egyszerűbb kód kedvéért, azért nem sorolom itt fel)*
- *Map<String, FormComponent> components*
  - *Ebből a űrlap-mező gyűjteményből majd elő lehet állítani az új rekordot..*

abstracts/ViewBuilder.java

A legfontosabb szerepe abban van, hogy felépítse az adott nézetet mihelyst odanavigálunk egy menu-gombbal. A *switchToTargetView()* fügvény akkor fut le, amikor erre nézetre lett navigálva egy gombbal, ami lecsereli az ablak tartalmáz a *base* JPanelre, ami itt épült meg az abstract *addComponentsToPanel()* metódussal. (csak egyszer, miután először lett megnyitva az adott nézet)

Attribútumok:

- *JPanel base*
  - *Erre a JPanelre fog mindent rápakolni*
- *MainFrame frame = MainFrame.getInstance();*
  - *Csak ide van hivatkozva, az ablak példánya (mert az singleton)*

abstracts/TableViewBuilder

A *ViewBuilder* leszármazottja, táblázatos nézetre specializálva.

(Új)Attribútumok:

- *Map<String, FormComponent> attributeComponents:*
  - *Ez tárolja az űrlap-elemket, attribútum névvel meghivatkozva. Ennek az osztálynak a leszármazottjainak a feladat ezt elkészíteni a létrehozott komponensekből*
- *CRUDTableModel tableModel*
  - *A JTable tablemodelje, ez a felüldefiniálás csak annyit tesz, hogy kell lenni addRow és removeRow függvényének pluszban.*
- *JButton addButton*
- *JButton deleteButton*
  - *Minden táblázatos nézet tartalmaz ilyen gombokat, ezért általánosítottam.*
- *JTable table*
  - *A leszármazottak feladata előállítani a táblázatos, amely a fenit modelt fogja használni.*
- *AbstractCRUDTableController controller*
  - *Majd ez lesz a gombok ActionListner, ha kész van a nézet felépítése akkor csak példányosítani kell, úgy, hogy a paramétere az aktuális TableViewBuilder (“new TableViewBuilder(this)”)*

További fontos osztályok

commons/Storage.java

Singleton, ez végzi a szerializálást. Ennek a példánya tárolja az összes betöltött adatot.A public void *loadAllFromFile()* metódus mindent beolvas a JSON fájlokból, a *saveAllToFile()* mindent kiír.

Attribútumok:

- *List<Shepherd> shepherds;*
- *List<Company> companies;*
- *List<Animal> animals*;

commons/MainFrame

Singleton, JFrame leszármazott. Az ablakot működteti

commons/MenuButtonListener

Ez működteti a gombos menürenszeremet, a menügombok *actionListener*je A gombra kattintáskor meghívja a kapott *ViewBuilder switchToTargetView* függvényét, ami felépíti a célnézet kinézetét.

Attribútumok

- *ViewBuilder viewBuilder*
    - *Konstruktorparaméter.*

Továbbiak

A további osztályok nagyrészt az előzőek leszármazottjai. Mind a 4 entitás típusomhoz *(Shepherd, Animal, Company, ShepherdAnimal*) implementálva van egy:

- **TableViewBuilder** leszármazott
- Egy **AbstractCRUDController** leszármazott
- Egy **CRUDTableModel** leszármazott
- És maga az entitás modell (pl. **Shepherd**.java)

A többi osztály általában valamilyen segédosztály (*ContentValidator, MenuButtonListener,..*), kisebb és egyszerűbb működést hajt végre, azokat itt nem részletezem. De a forráskódban dokumentálva vannak ezek is.

Adatszerkezetek

Három JSON fájlban tárolom az adatokat (animal.json, shepherd.json, shepherd.json), melyek szerkezete szinte teljesen megegyezik az adatokéval, annyi különbséggel, hogy a kapcsolatokat (pl. *Shepherd.getEmployer()*) id-n keresztül tárolom el. Kiíratáskor a referált objektum általam definiált id-ját tárolom el, és visszaolvasáskor ez alapján 'bereferálom' az objektumot.

Pl.egy részlet a *shepherds.json*-ből:

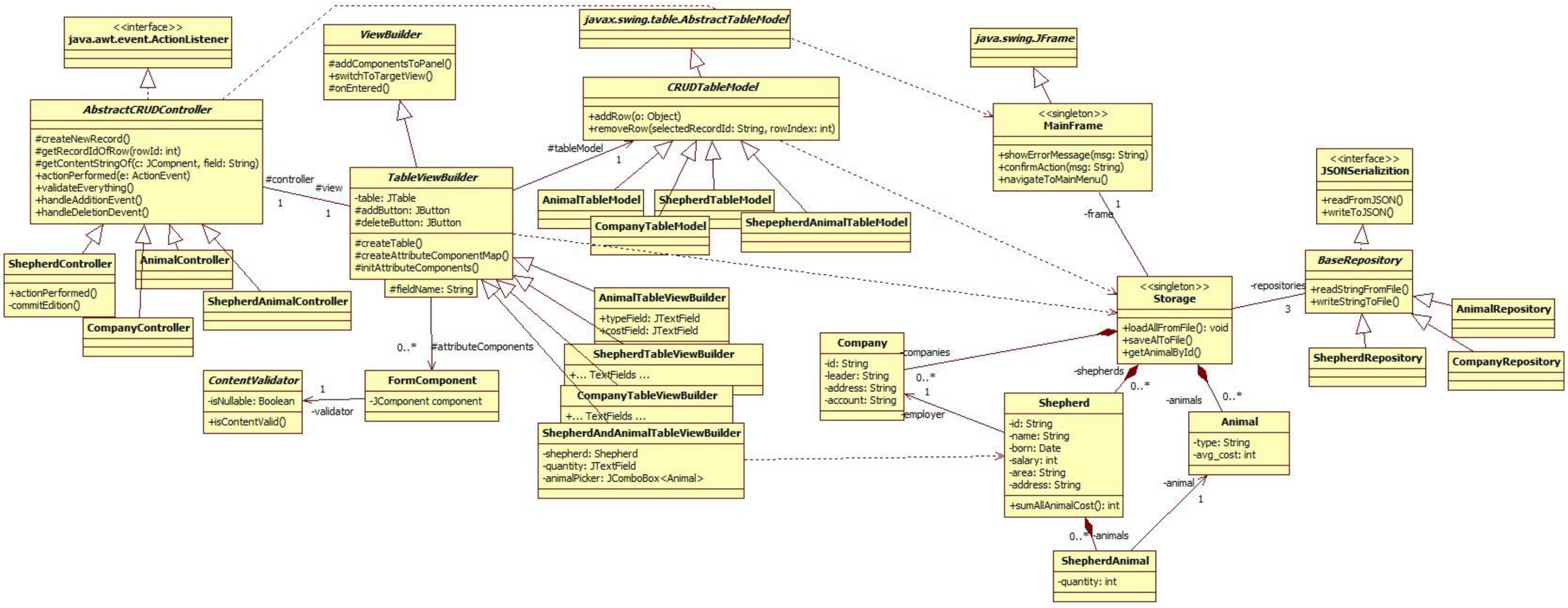
```
"name": "Teszt Tamás",
"animals":[{"quantity":5,"animal":"2d8f647f4e854a119ee976cbbc415658"}],
"salary":190000,…. }
```

Betöltéskor itt, az animal-nál az *id*-helyett az *id*-által referált *Animal* példány (pointere) került be. Az itt látható kódrészlet megmutatja ezt (másolva: *commons/ShepherdRepository/getShepherdAnimal()*)

```
List<ShepherdAnimal> result = new ArrayList<>();
    for (int i = 0; i < animals.length(); i++) {
      JSONObject actual = animals.getJSONObject(i);
      result.add(
new ShepherdAnimal(Storage.getAnimalById(actual.getString("animal")),actual.getInt("quantity"))
);
    }
```

Ugyanígy működik az *employer* 'bereferálása' a *shepherd*-hez

Osztálydiagram



A gettereket és settereket elhagytam, a könnyebb érthetőség kedvéért, A főbb működés szempontjából nem releváns metódusokat, és osztályokat elhagytam, hogy a diagram mérete ne nőjön (*ennél is*) nagyobbra.

Kimaradt osztályok:

- *MainMenuViewBuilder*
  - *Fömenüt összerakó osztály, ViewBuilder leszármazott*
- *MenuButtonListener*
  - *(lényege feljebb leírva)*
- *VisualizationViewBuilder*
  - *A diagramrajzolást valósítja meg, a GraphDrawert használva*
  - *ViewBuilder leszármazott*
- *GraphDrawer*.
  - *Oszlopdiagramok rajzolására készített segédosztály*

Fontos metódusok működésének leírása

A legtöbb metódus **kommentelve van** JavaDoc-ként (*kivéve az abstract osztályok leszármazott osztályainak örökölt metódusai, mivel azokra értelemszerűen az ősosztály kommenjei vonatkoznak. De ahol lényeges különbség van az ott leírttól, akkor azt megmagyaráztam*)

Itt csak pár, szerintem legfontosabb általános metódus belső működéséről írok, valamint a diagram rajzolásról.

Validálás folyamata

Abstracts/AbstractCRUDController/validateEverything

```
boolean everythingIsValid = true;
    for (Map.Entry<String, FormComponent> entry : this.components.entrySet()) {
        FormComponent c = entry.getValue();
        if (!c.getValidator().isContentValid(this.getContentStringOf(c.getElement(), entry.getKey()),
c.getFieldName())) {
            everythingIsValid = false;
            break;
        }
    }
return everythingIsValid;
```

Az itt látható kódrészletben végigiterálok a az adott nézet form komponentsein (*JTextField és JComboBox-ok*) és a megadott *ContentValidator* osztályok segítségel a komponensek tartalmát leellenőrzi. Amennyiben egy is rossz, azt jelzi egy *JOptionPane*-es üzenettel (amint a *isContentValid* metódusjeleníttet meg.). Ilyenkor a hozzáadási folyamat megszakad.

Hozzáadaás folyamata

Abstracts/AbstractCRUDController/handleAdditionEvent

```
protected void handleAdditionEvent() {
    if (this.validateEverything()) {
        this.tableModel.addRow(this.createNewRecord(this.components));
        this.setEverythingToEmpty();
    }
}
```

Az itt látható módon, a validálást követően elég a *TableModel-lel* hozzáadatni a rekordot, ami a *createNewRecord* metódussal készül (ami *abstract*), a *FormComponent map* rakják össze a leszármazottak a rekordot. A hozzáadaást követően pedig minden mező tartalmát üresre állítok.

Diagram rajzolás

Ezért a *visualization/VisualizationViewBuilder/GraphDrawer* osztály felel, a megadott adatlistából előállítja az oszopdiagrammot, ami úgy történik, hogy elöszőr normalizálja az adatokat 0 és a berajzolható terült magasságának értéke közé, ami így történik:

visualization/VisualizationViewBuilder/GraphDrawer/NormalizeHeightValues

```
values.stream().map(
i -> (int) Math.round(this.getGraphHeight() * ((i - min) / (double) (max -min)))
).collect(Collectors.toList());
```

Vagyis labda kifejezést használva, minden értéket az előbbi képlettel átváltok arra a tartományra, melyben ábrázolni tudom az oszlopokat, a *Graphics* osztály függvényei segítségével.

Az osztály továbbá kiszámolja, hogy milyen szélesek legyen az oszlopok, és közöttük mekkora távolság legyen, és ez alapján a berajzolja őket különböző színtkkel kiszínezve.

Tesztek leírása

Igyekeztem a legtöbb értelmesen tesztelhető funkciót tesztelni:

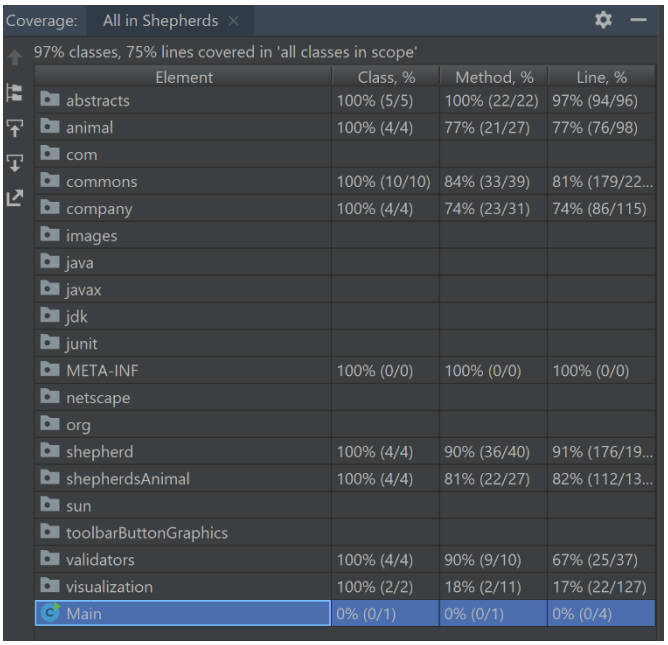
Összesen 12 tesztesetet írtam: Minden entitás típus hozzáadására és törlésére, és a szerializálás típusonkénti ellenőrzésére.

Tehát a tesztek a következők:

- *addAnimalTest, deleteAnimalTest*
- *addCompany, deleteCompanyTest*
- *addMinimalShepherdTest, addFullShepherdTest*
    - *a 'Minimal' verzió csak a két kötelező attribútummal való hozzáadást teszeli, még a 'Full' verzió minden lehetséges adattal (vagyis készít hozzá employer Companyt is)*
- *editShepherd, deleteShepherd*

    - *(‘edit’-test azért csak a shepherd-nél van, mivel a többi résznél a JTable-lel lehet végezni a módosítást.)*
- *addAndThanDeleteAnimalFromShepherd*
    - *Létrehoz egy pászort minden adattal, majd létrehoz egy állatot és hozzáadja és törli a pásztortól.*
- *AnimalSerializationTest, CompanySerializationTest, ShepherdSerializationTest*
    - *Itt hozzáadok rekordokat, majd beleírom a fájlba és utána visszaolvasom a fájlból, és ezt tesztelem.*
    - *Ezek a tesztek külön, ‘…_test.json’ fájlokkal dolgoznak, hogy ne legyen összekavarodás.*

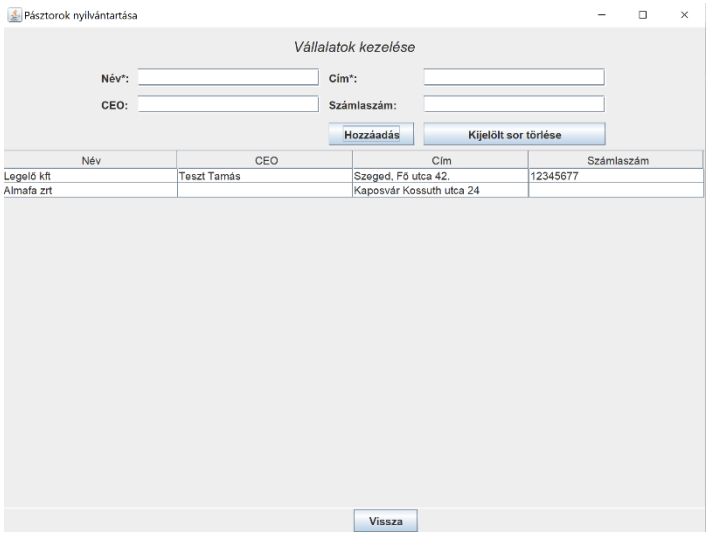
Az alábbi képen látható a tesztlefedettség *(IntelIJ-vel dolgoztam)*



Felhasználói kézikönyv

A programot elindítva 4 gomb fogad, melyek a különböző entitást-típusok módosító-kezelő mezőihez irányítanak, valamint az *Visualization* című gombbal pásztorok adatait lehet vizualizálni.

Állatok és vállalatok kezelése

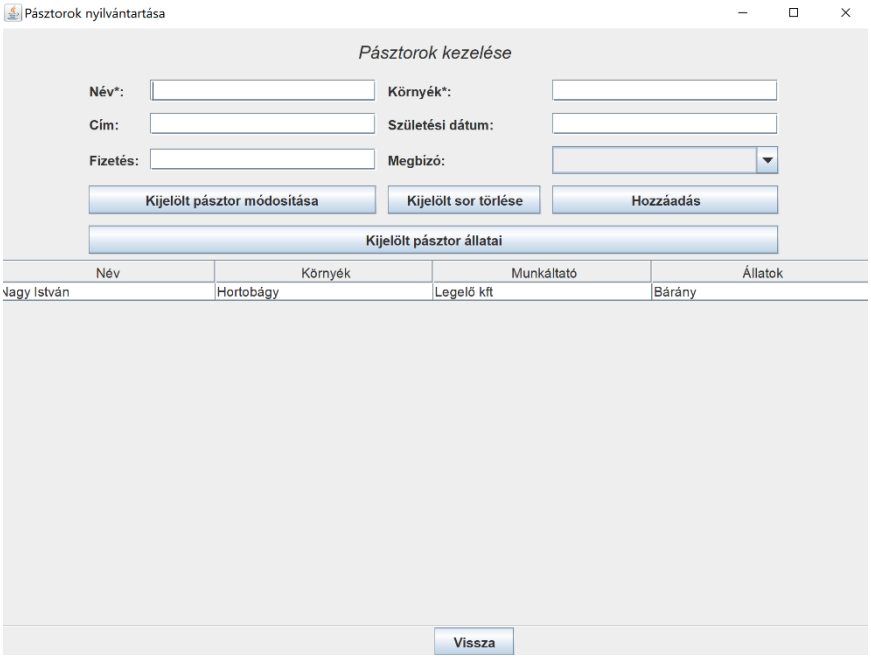


Az itt látható táblázatok a fent látható beviteli mezőkön lehet beállítani az értékeket, a *-gal jelöltek kötelezőek. Az adatok helyességét ellenőrzi a szoftver, hibás esetben figyelmeztet. Módosítani a táblázaton kiválasztott cellába kétszer kattintva lehet.

A táblázat fejlécén kattintva lehetőség van a rekordok rendezésére is.

Egy sor lehet kijelölni, majd a törlés gomra kattintás után egy megerősítéssel lehet törölni.

Pásztorok kezelése



Ez annyiban különbözik az előzőkeltől, hogy módosítani nem a táblázatban a cellákra kattintáskor van lehetőség, (mivel itt nem férne ki az összes attribútum) hanem a sor kiválasztása után a módosítás gombra való kattintáskor az adott rekord összes adata betöltődik a fenti űrlap mezőkbe.

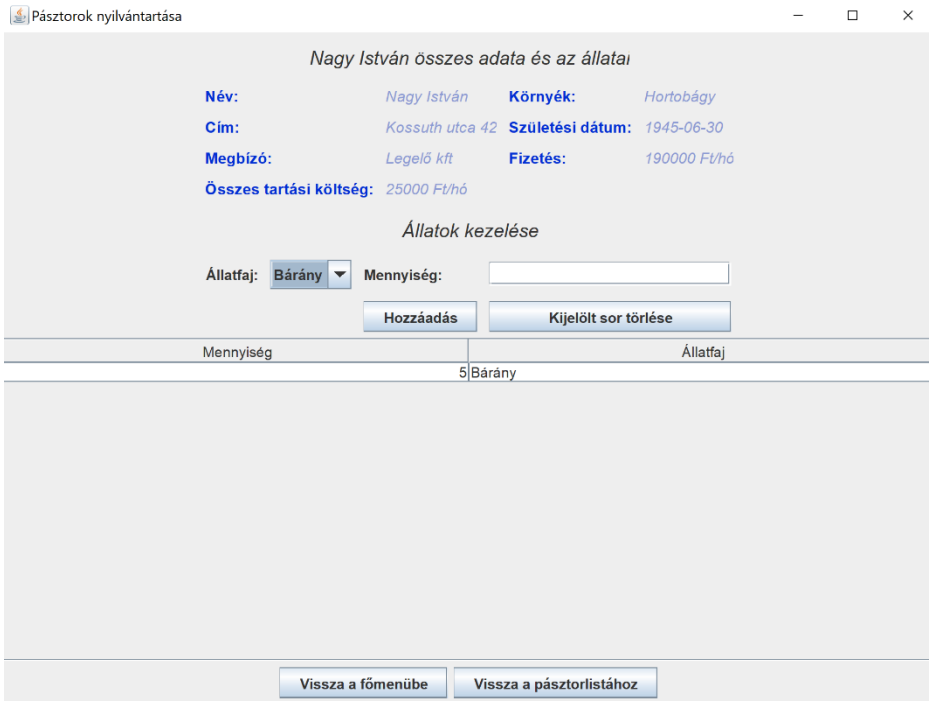
A 'Kijelölt pásztor állatai' nevű gombra kattintva megjelenik az állat adminisztráló nézet, a kiválasztott pásztorhoz.

Pásztor állatainak kezelése

Az itt látható nézetben lehetőség van állatokat felvenni az adott pásztorhoz. Módosítani itt csak mennyiséget lehet, és ezt a cellamezőbe kattintással lehet megtenni.

Ha fel szeretnénk venni egy olyan állatot az adott pásztorhoz, ami már létezik nála, akkor nem kerül új sor beszúrásra, hanem csak a meglévő sorban levő mennyiséget fog megnövekedni.

Az összes tartási költséget megadó mező egy számított mező, csak a nézet újra betöltődésekor frissül.



Adatvizualizáció

Itt a bal oldalon lévő listából kiválasztott pásztorok adatait lehet összehasonlítani.

Három tulajdonság alapján lehet oszlopdiagrammot készíteni:

1. Pásztor által birtokolt állatok összesített tartási költsége
2. Pásztorok fizetései
3. Pásztorok állatainak száma

Nem összefüggő intervallumot a CTRL nyomva tartásával lehet kiválasztani.Összesen maximum 10 pásztort lehet bevenni egy ilyen összehasonlításba.