Teszári Gergely (UBMX0L) – Objektumorientált programozás Nagy házi feladat
Felhasználói és programozói dokumentáció

Jelen dokumentáció egyaránt tartalmazza a felhasználóknak szánt használati utasítást, a programozóknak / továbbfejlesztőknek szóló részletes programleírást, valamint a programban lévő osztályok UML-diagramját.

Tartalom

Felhasználói dokumentáció

Üdvözli a Jaser telefonkönyv program felhasználói útmutatója! Ebben az átfogó leírásban megismerheti a program helyes használatát, funkcióit. Kezdjünk is bele!

Grafikus felület:

A program egy konzolos ablakban jelenik meg. Első indításkor nem áll másból, mint a programból. (.JAR futtatható Java fájl, vagy Windows executable .EXE) A program automatikusan adatbázist keres, és ha nem talál a saját elérési útján adatbázis fájlt, akkor adatok nélkül is képes elindulni, korlátozott funkcionalitással. Ha talál adatbázist, az alábbi menü fogad minket. Ez a menü minden operáció után vissza fog térni.

```
Üdvözlöm a Jaser telefonkönyv programban!
Mit kíván tenni?

Adatok bevitele:        b

Adatok törlése:         t

Adatok listázása:       l

Keresés:                k

Adatok exportálása .VCF fájlba: v

Mentés: m

Mentés és kilépés:              x

>>>
```

Új hozzáadása

Amennyiben még nem adtunk hozzá egyetlen kontaktot sem, megtehetjük a „b” parancs kiadásával. Ekkor a program egyesével bekéri az új kontakt adatait. A több elemű vezeték - vagy keresztneveket szóközzel elválasztva megadhatjuk.

```
Adja meg az új rekord vezetéknevét:
>>>
Minta

Adja meg az új rekord keresztnevét:
>>>
Sándor

Adja meg az új rekord becenevét:
>>>
Sanyi

Adja meg az új rekord lakcímét:
>>>
-

Adja meg az új rekord munkahelyi telefonszámát:
>>>
06300000000

Adja meg az új rekord otthoni telefonszámát:
>>>
06301234567

Sikeres hozzáadás!

Üssön entert a folytatáshoz...
```

Adatok kiíratása

Ha ezek után kíváncsiak vagyunk a bevitt rekordokra, a főmenübe visszatérve adjuk ki az „l” parancsot:

```
Keresztnév: Minta
Vezetéknév: Sándor
Becenév: Sanyi
Cím: -
Munkahelyi telefonszám: 06300000000
Privát telefonszám: 06301234567
==========================


Üssön entert a folytatáshoz...
```

Törlés

Ha valamilyen oknál fogva törölni szeretnénk egy kontaktot a listából, arra használjuk a „t” paracsot! Ekkor a program kiírja a létező kontaktokat, és telefonszám alapján törölhetünk:

```
Keresztnév: Minta
Vezetéknév: Sándor
Becenév: Sanyi
Cím: -
Munkahelyi telefonszám: 06300000000
Privát telefonszám: 06301234567
==========================

Adja meg a telefonszámot, amihez tartozó kontaktot el kívánja távolítani:
>>>
```

Ha létező telefonszámot adunk meg, a sikeres törlést a program vissza is jelzi:

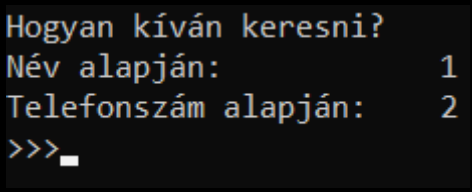
```
Keresztnév: Minta
Vezetéknév: Sándor
Becenév: Sanyi
Cím: -
Munkahelyi telefonszám: 06300000000
Privát telefonszám: 06301234567
==========================

Adja meg a telefonszámot, amihez tartozó kontaktot el kívánja távolítani:
>>>06300000000
Sikeres törlés: 06300000000

Üssön entert a folytatáshoz...
```

Keresés

Szintén nagyon hasznos funció a keresés. Ezt a „k” paranccsal érhetjük el. Ilyenkor a program egy kisebb almenüt mutat, amiben kiválaszthatjuk, hogy hogyan szeretnénk keresni:



Ha név alapján kívánunk keresni, („1”-es parancs) akkor kereshetünk teljes névvel, vagy a név egy darabjával. A program minden - a megadott névvel egyező nevű - kontaktot ki fog írni a kijelzőre.

Ha vezetéknév - keresztnév párost adunk meg, azt szóközzel elválasztva tegyük, különben a program egy névként értékeli. A vezetéknév és a keresztnév sorrendje nem fontos. Ezen felül a becenévvel is képes megtalálni a kontaktokat, az előbbiekhez hasonlóan. Ha a becenév bárhol szerepel a beírt névben, akkor a kontaktot 15 féle képpen megtalálja a program.

A telefonszám alapú keresésnél („2”-es parancs) a program megtalálja mind a munkahelyi, mind a privát számok között a keresett számot.

VCF fájl

A **.VCF** fájlba írás rendkívül hasznos, mivel ezt a fájlt a legtöbb Androidos telefon és még a Windows is képes beolvasni és értelmezni. Ha szeretnénk az adatbázist .**VCF** fájlba kiíratni, adjuk ki a „v” parancsot! Ekkor a program visszajelez, ha a fáljba írás sikeres volt.

Mentés / mentés és kilépés

Ha végeztünk teendőinkkel, vagy csak azt szeretnénk, hogy mentsen a program, ne a jobb felső sarokban található X-el lépjünk ki, használjuk az „x” parancsot a kilépéshez, az „m” parancsot a kilépés nélküli mentéshez. Ekkor a program menti az adatállományát egy .**TXT** fájlba, ennek sikerességéről értesít, majd ezután kiléphetünk. Ha a következő indításkor ez a .**TXT** fájl egy helyen van a program futtatható fájljával, akkor beolvasva onnan folytathatjuk a munkát, ahol abbahagytuk.

Fontos tudnivalók

- A program a **.TXT** adatbázis nélkül mindig újonnan indul. Ha nem találja maga mellett ugyanazon a helyen a fájlt, akkor üres tárolóval indul.
- Ha még nincsenek adatok a rendszerben, akkor a listázás, keresés, törlés és **.VCF** exportálás funkciók ki vannak kapcsolva. Ez érvényes akkor is, ha minden adatot kézzel kitörlünk.
- A **.TXT** adatbázist kézzel ne módosítsuk, mert ez a program hibás működéséhez vezethet.
- A program **.VCF** fájlt beolvasni nem képes, csak kiadni tudja a már tárolt adatokat.
- A több tagú vezeték - és keresztnevet a program külön nem kezeli. Többtagú nevek megadhatóak bekéréskor szóközzel szeparálva. Esetleg érdemes a harmadik nevet a becenév mezőben megadni.
- Egy telefonszám pontosan egyszer szerepelhet az adatbázisban. Ha új felvételekor olyan telefonszámot adunk meg, ami már szerepelt korábban, akkor a rendszer figyelmeztet, és nem engedi hozzáadni az új rekordot az adatbázishoz.

PROGRAMOZÓI DOKUMENTÁCIÓ

Ebben a programozói dokumentációban fájlonként megyünk végig a programon. Nyolc .java fájlból áll a program, ezek a funkcionalitásuk szerint szétválogatott metódusokat tartalmaznak. Ezek sorban: Main.java, DataManipulation.java, FileOperations.java és az Entry.java.

MAIN.JAVA

Kezdjük a **Main.java** fáljllal, ami az egész program main osztálya. Ebből indul a program, ebbe tér vissza újra és újra. Legelőször a statikus Scanner input változót deklarálja, ez nem kerül bezárásra, később láthatóan a végtelenségig ismétlődik egy menüben. Ezért is static. Két kis metódus, segéd metódus is megelőzi magát a fő attrakciót: az **exit()**, és a **cls().** Az **exit()** mindössze annyit tesz, hogy meghívásakor vár egy billentyű lenyomására, majd Enterre. Ez segít abban, hogy a kiírt eredmények láthatóak legyenek, ne tűnjenek el azonnal, illetve a program bezárásakor legyen egy megerősítés. A **cls()** feladata a konzol tisztítása, esztétikai okokból, az áttekinthetőség segítésére. Ez kizárólag Windows-on működik, mivel a Windows beépített konzol törlő utasítását adja ki futása során.

Ezek után következik a **main()** metódus, ami a végtelenszer ismétlődő menü kiírás. Először létrehoz egy ArrayList-et, ami Entry típusú adatokat fog tárolni, majd meghívja rá a **FileOperations.fileInput()** metódust, hogy töltse fel adatokkal, fájlból, ha tudja. (Bővebben erről a metódusról később.) Innentől indul a végtelen loop, a while(true), ami a következőképpen épül fel:

- Először mindig tisztítja a konzolt **(cls())**
- Utána kiírja, hogy mit vár, mi ez a program. Üdvözli a felhasználót, felsorolja neki a funkciókat, és hogy milyen karakter beütésével érheti el őket, majd megnyitja a Scanner inputot, ahol az utasítást várja. Fontos megjegyezni, hogy mielőtt bármit is kiírna, ellenőrzi, hogy az ArrayList hossza nem nulla, azaz beolvasott-e bármit fájlból, amivel fel tudta tölteni a listát. Ha a lista üres, csak a bevitelt jeleníti meg, mint lehetőség.
- A lehetőségek egy else-if többutas elágazás sorozatban kerülnek lekezelésre, minden esetben a kisbetűssé konvertált betűt méri össze a lehetőségben megadott betűvel. Ha egyik lehetőséggel sem egyezik a beírt érték, akkor minden ezektől eltérő esetben hibát jelez, és kezdi előről a while ciklust.
- A lehetőségek közül a listázás, keresés, törlés, VCF fájlba exportálás mind ellenőrzik, hogy nem üres-e a lista, ebben az esetben csak hibaüzenetet írnak ki, nem hajtják végre az utasítást akkor sem, ha a nem megjelenített karaktert mégis beírja a felhasználó. Ez alól kivétel a mentés és a mentés és kilépés opciók, ezeket lehet úgy is, hogy nem írta ki a lehetőségét a program.
- Különben az opciók így működnek:
    - Adatbevitel, törlés, keresés: függvényt hívnak a DataManipulation osztályból
    - VCF export, mentések: függvényt hívnak a FileOperations osztályból
    - Adatok listázása: Foreach ciklussal kiírja az Entry.toString() metódus segítségével az adatokat a konzolra.
- Minden operáció konzol törléssel kezdődik, és az exit() hívásával fejeződik be.
- A metódusok tetszelés - kompatibilitása miatt a hívott függvényekben minimális a felhasználói interakció, így az adatok bekérése, eltárolása, a hívott függvények futási módjának meghatározására a **main()** hívogatja a **DataManipulation.GUI()** metódust.

Ezzel a Main classot befejeztük, térjünk át az **Entry.java**-ra.

ENTRY.JAVA

Ez az osztály az adattag saját osztálya. Az adattag neve Entry, komponensei pedig:

- lastName
- firstName
- nickname
- address
- workNum
- privateNum

Ehhez megfelelően definiálva van a konstruktor, getter - setter metódusok, ahogy kell, ez ennél részletesebb magyarázatot nem igényel. Vannak viszont itt más metódusok, amik igen. A **toString()** metódust felüldefiniálva használjuk a képernyőre iratáshoz. Ez az adattagokat szép sorban, egymás alatt jeleníti meg, mind elé odaírja, hogy mi az, magyarul, kettőspont, majd az adattag, az egész blokk felett és alatt sortörés, plusz alul egy sor egyenlőségjel is található, a még jobb vizuális szeparáció érdekében. Két kiirató metódus van még: a **toFile()** és a **toVCF**(). A fájlban alulhúzással elválasztva írjuk ki az adattagokat, meghatározott sorrendben, ezt használjuk a txt fájlba írt adatbázishoz. A VCF fájlba írása már trükkösebb. Kellenek a megfelelő szintaktiai elemek, megfelelő karakterek, sortörések, és egy plusz is, hogy a magyar ékezetes betűket is tárolni tudjuk. Ehhez külön metódusban van a **toHexASCII(),** ami megkap egy szót, aminek karakterein egyesével végigmegy, minden karakter hexadecibális ASCII kódját hozzáfűzi a visszatérési stringhez. Szeparáló karakternek az egyenlőségjelet használja.

EntryData, EntryName, EntryAddress, EntryPhone

Az **EntryData** az Entry adattagjait tároló típus, melyből származnak az **EntryName**, **EntryAddress** és az **EntryPhone** osztályok. Itt jelenik meg érdemben az OOP elve. Értékei Stringek.

DataManipulation.java

Ez az osztály felel minden adatmódosításért, amit az adatbázison végzünk. Itt is van elsőnek egy statikus Scanner input változó, csak úgy, mint a mainben. Ebben is van egy kismetódus, az **isInTheList().** Ez végig megy a neki átadott ArrayList-en, ami Entry-ket tartalmaz, és ha az aktuális elem munkahelyi vagy privát száma megegyezik a szintén megkapott telefonszámmal, akkor visszatér a referenciájával, különben folytatja, és ha végigért, és nem talált egyezést, akkor visszatér none-al.

Az első érdemben nagyobb metódus a **GUI(),** ez végzi a különböző adat bekéréseket a main() által hívott függvényekre. Erre azért van szükség, hogy a sok adatbekérés ne tegye túl hosszúvá és átláthatatlanná a maint, és hogy a függvények tesztelhetőek legyenek. Esetszétválasztást végez, ami első része, hogy ha üres az adatbázis, akkor nem engedi a listázás, törlés, keresés, VCF export függvényeket hívni. Ha ez a feltétel nem teljesül, vagyis van valami az adatbázisban, akkor a keresésnél bekéri a keresési módot és értéket, új hozzáadásnál felveszi az új adatokat, összeállítja az Entry-t belőle, törlésnél kiírja a lista tartalmát, és bekéri a törölni kívánt telefonszámot.

Ezt követi az **addNew(),** ami - ahogy a nevéből is adódik – az új kontakt hozzáadását végzi az adatbázishoz. Minden adatot sorban megkap a GUItól, és mivel megkapja az adatbázist is, az **isInTheList()**-tel megvizsgálja, hogy a telefonszámok közül szerepelt-e már bármelyik korábban, és ha igen, hibaüzenet kíséretében nem adja hozzá a listához. Ha a telefonszámok egyike sem szerepelt még sehol korábban, akkor hozzáadja a listához. Fontos megjegyezni, hogy a telefonszámot nem számként tárolja a program, hanem stringként, hogy a +36-os számokat, a kötőjellel szeparált és egyéb számokat is képes legyen tárolni.

A következő metódus a **removeEntry().** Ez a GUIból megkapja a telefonszámot, amit törölni kívánunk, az **isInThe List()**-tel megnézi, hogy létezik-e, és ha nem, hibaüzenetet dob. Ha igen, akkor az isInTheList-től kapott referenciájú elemet eltávolítja az ArrayList-ből.

Talán a legösszetettebb metódus a **searchEntry().** A GUIból megkapja az adatbázist, hogy hogyan szeretnénk keresni, (név vagy telefonszám alapján) és a keresési értéket. Ha név alapján keresünk, megpróbálja splittelni a szóközök mentén. Ez után végigmegy az ArrayList-en, és a soron lévő Entry név adatait összefűzi egy szóközökkel elválasztott stringgé. Ezt követően végigmegy a kapott (és esetleg splittelt) adaton, és ellenőrzi, hogy tartalmazza-e a darabo(ka)t az összefűzött soron lévő elem az adatbázisból. Ha igen, hozzáadja a visszatérési ArrayList-hez, és továbblép, hogy ha több elem is egyezne, ne adja hozzá többször. Ha nem, továbblép, végül, ha üres maradt a lista, hibaüzenetet ad.

Ha telefonszám alapján akarunk keresni, akkor jóval egyszerűbb a helyzet, mivel az **isInTheList()**-tel megnézi, szerepel-e egyáltalán ilyen telefonszámmal adat, és ezt egy ideiglenes változóban eltárolja. Tehát az **isInTheList() visszatérési** értéke lesz a válasz, ami ha null, akkor nincs ilyen telefonszámmal adat, ezt hibaüzenettel jelzi is, vagy pedig a megfelelő Entry referenciája, amit hozzáad a visszatérési ArrayList-hez. Végül mindig visszaadja a visszatérési ArrayList-et, ami, ha üres, akkor nincs találat, ekkor kiírásra sem kerül a GUIban.

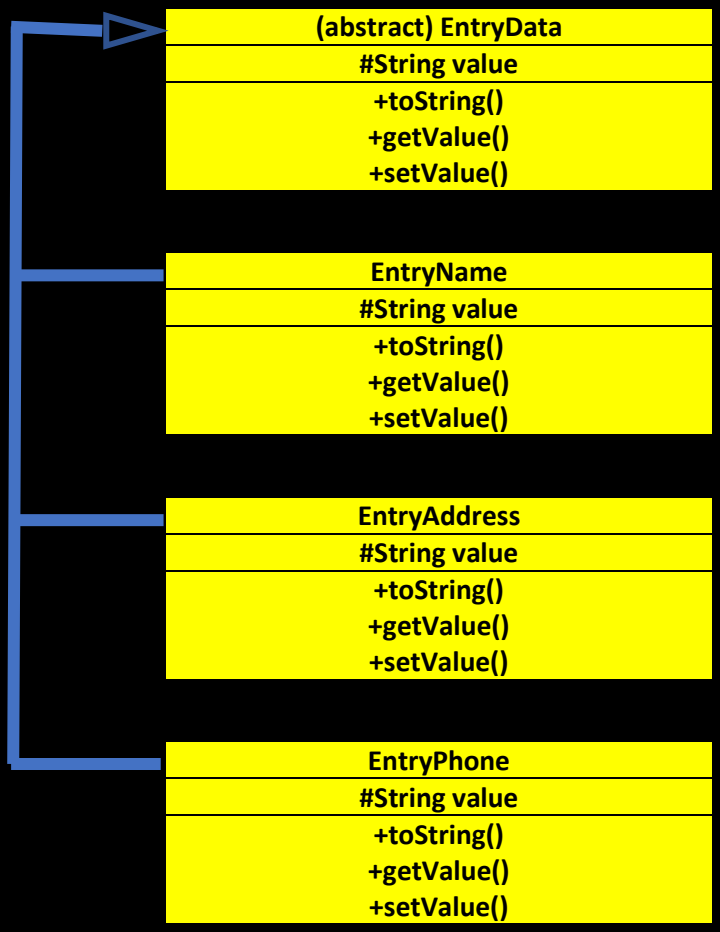
FileOperations.java

Ezzel el is érkeztünk az utolsó osztályhoz, ami a fájlműveletekkel foglalkozik. Kettő metódusból és egy mini metódusból áll. A kicsi a **stringToEntry(),** igazából kap egy String[] tömböt, amit átalakít egy Entry-vé. A **fileInput()** a **Main.main()** által először és egyszer hívott függvény. Ez felelős az Entries.txt fájl beolvasásáért, sorokra tördeléséért, a sorok alulhúzások mentén splitteléséért, és az így kapott String[] tömbökből a **stringToEntry()** segítségével Entry-k készítéséért. Ezeket az objektumokat aztán berakja egy Entry típusú ArrayList-be, amivel - ha végzett a fájl olvasásával - visszatér. Kezeli a nem létező fájl hibáját, és a fájlban esetlegesen nem szétdarabolható hibás dolgokat is.

A **fileOutput()** metódus megkapja a fájlba írandó listát, valamint a kimeneti fájl formátumát. Ha .txt a kért kiírási módszer, akkor az **Entry.toFile()** metódust használja, ellenkező esetben az **Entry.toVCF()**-et. Végül tájékoztat az írás sikerességéről vagy sikertelenségéről.

Ezzel végigvettük a teljes program felépítését.

UML diagramok



Entry
-EntryNamelastName; - EntryNamefirstName; - EntryNamenickName; - EntryAddress address; - EntryPhoneworkNum; - EntryPhoneprivateNum;
+String toString() +String toFile() +String toVCF() +String stringToHexASCII() +String getlName () +String getfName () +String getnName () +String getwNum() +String getpNum()

Main
-Scanner input
+exit() +cls() +main()

DataManipulation
+Scanner input
+isInTheList() +GUI() +addNew() +removeEntry() +searchEntry()

FileOperations
-
+fileInput() +stringToEntry() +fileOutput()