Dokumentáció

A program környezete és célja:

Szimulálja egy gyümölcs kereskedő (a program felhasználója) világát. Ahol 7 féle gyümölcs létezik, a kereskedő farmerektől vásárolhat gyümölcsöt, majd azokat profitért eladja a piacon a vásárlóknak. A felhasználó ebben a helyzetben választhat adminisztrációs parancsokból például: Automatikus kereskedés a farmerekkel. Ezekkel a parancsokkal változtatja a környezetét, például, hogy kikkel kereskedhet, mennyi pénze van amivel kereskedhet. Minden farmer egy féle gyümölcsöt termeszt.

A program a környezet történelmét rögzíti: a tranzakciók információit, illetve a rendelkezésre álló farmereket.

A program felépítése:

Egy menü fogadja a felhasználót 9 opcióval:

* Egyenleg megtekintése: a felhasználó megtekintheti egyenlegét. Ez az első futtatáskor 1000.00 és ahogy a felhasználó kereskedik nő az egyenlege.
* Farmerek listájának megtekintése: a felhasználó megtekintheti a farmereket (tőlük vásárolhatja meg a gyümölcsöt, amit aztán eladhat profitért).
* Gyümölcsök tuldajdonságainak megtekintése: a felhasználó megtekintheti milyen tulajdonsággal rendelkeznek az ebben a virtuális világban szereplő gyümölcsök, ezek teremhetnek a farmereknél és ezekkel folyik a kereskedés.
* Nyugták listájának megtekintése: itt tekintheti meg a felhasználó a múltbéli tranzakcióit.
* Farmer hozzáadása a listához: a farmer ezzel az opcióval adhatja hozzá farmer ismerőseit, akiktől beszerzi a gyümölcsöt. Meg kell adnia 3 adatot egy farmer előállításához: nevet, termesztett gyümölcsöt és kívánt napi keresetet. Azért a napi keresetét adja meg a farmernek, mert a farmer eldönti hogy ő mennyit akar keresni egy nap alatt a gyümölcsei eladásából. Ezt a felhasználóval megbeszéli (gyakorlatilag nem, mert az egész fiktív, de ezt a szituációt feltételezve íródott a program), a felhasználó pedig amikor a farmereiről szóló adatbázist készíti ezzel a menüponttal, beírja.
* Farmer eltávolítása a listából: ezzel az opcióval törölheti a farmerei közül a felhasználó az egyiket. Például ha kukacos almát adott neki és legközelebb nem bízik meg benne, akkor csak törli a kereskedelmi köréből. Ezt úgy teheti meg, hogy beírja a törölni kívánt farmer nevét.
* Farmer tulajdonságának módosítása: ezzel az opcióval módosíthatja a farmer egyik tulajdonságát. Meg kell adnia a farmer nevét, majd az almenüben kiválasztani melyik tulajdonságát kívánja módosítani, ott beírja az új adatot a kiválasztott tulajdonsághoz és kész.
* Automatikus kereskedés a farmerekkel: ezzel az opcióval kereskedhet a felhasználó. Itt nagyon sok minden történik, de a felhasználó nem érzékeli, ő csak kiválasztja ezt az opciót és a program kereskedik neki, arról, hogy hogyan beszélek később a megvalósításnál. A lényeg a felhasználónak, hogy ez a menüpont vesz neki a farmerektől és elad a piacon profitért annyi gyümölcsöt, amennyit csak lehet, így növelve a felhasználónk egyenlegét.
* Program bezárása (Automatikus mentéssel): ezzel az opcióval zárja be a felhasználó a programot.

Megvalósítás

Program tartalma:

* 4 osztály:
  + Main, ezt futtatja a felhasználó amikor elindítja a programot, itt találhatóak a függvények.
  + Farmers, ez az osztály tartalmazza a farmer példányok tervrajzait (név, napi kereset, árult gyümölcs, mennyiség(ezt véletlenszerűen generálja)),, eladás napja(ezt véletlenszerűen generálja)), itt található a konstroktoruk, a getter, setter függvényeik.
  + Fruits, ez az osztály tartalmazza a gyümölcs példányok tervrajzait(név, romlás mértéke(megadja, hogy hány százalékkal csökken naponta a vétel és az eladás között), piaci értéke 0 naposan(ez csökken minden nap a romlás mértékével, ez 0 lesz ha elég nap eltelik, a gyümölcs megromlásához)), ezek a Mainben static final-ként létre vannak hozva.
  + Receipts, ilyen típusban tárolja a tranzakciókat a program, van kereskedés előtti- és utáni egyenleg, profit és tranzakciók száma, ami azt határozza meg hogy a tranzakció során összesen hány alkalommal vásárolt és adott el 1 darab gyümölcsöt a felhasználó.
* 1 enum: fruitsNameEnum, tartalmazza a 7 gyümölcs nevét (Alma, Narancs, Kiwi, Szolo, Banan, Szilva, Korte).
* 2 xml file:
  + receipts.xml, ez tartalmazza, illetve ebből olvassa be a program a múltbéli vásárlások adatait.
  + farmers.xml, ez tartalmazza, illetve ebből olvassa be a program a rendelkezésre álló farmerek adatait.
* 3 static Arraylist: ezekben tárolja a program az előző vásárlás példányokat, a farmer példányokat, a gyümölcs példányokat. A Mainben vannak Inicializálva.
* 7 segédmetódus:
  + indexByFarmersName(), ez meghatározza a farmers Arraylistben a keresett farmer példány indexét név alapján.
  + indexByFruitsName(), ez meghatározza a fruits Arraylistben a keresett gyümölcs példány indexét név alapján.
  + fillFruitsData() ez a 7 static final Fruits típusú gyümölcsöt belerakja a fruits Arraylistbe.
  + marketProfit() a farmerek és a piac is random napokon történik, ez számolja ki, hogy mennyi lenne a profit egy darab gyümölcsön egy konkrét napon, a gyümölcs romlását figyelembevéve.
  + marketPrice ez számolja ki a piac napján a gyümölcs árát, ami 0 ha a gyümölcs teljesen megromlik a piac napjára.
  + createChild() az xml fájlok készítéséhez szükséges metódusokhoz kellett, csinál egy child property-t.
  + setYourCredit() kiszedi a receipts.xml fájlból az utolsó kereskedés utáni egyenleget és azzal teszi egyenlővé a felhasználó jelenlegi egyenlegét.
* 13 menüvel kapcsolatos metódus:
  + menu(): ez a metódus vizsgálja meg a felhasználó választását. Kér egy Integert 0-8 között és az alapján választja ki, hogy a felhasználó melyik menüpontot választotta. Ez egy while ciklusban fut, amíg a felhasználó nem a 8-as opciót (Kilépés és mentés) választja.
  + saveFarmersXml(): ezt a metódust nem tudja közvetlenül meghívni a felhasználó, hanem akkor lép életbe, amikor a felhasználó a mentés és kilépés funkciót választja. Menti a farmers Arraylist összes adatát a farmes.xml fájlba, documentBuilderFactoryt használ, egy for-each ciklussal megy végig a farmers-en, annyiszor ahány eleme a farmers-nek van.
  + saveReceiptsXml(): ugyanúgy működik, mint a saveFarmersXml(), annyi különbséggel, hogy a receipts adatait menti a receipts.xml fájlba.
  + walletMenu(): ha ezt választja a felhasználó, akkor kiírja a yourCredit nevű Double típusú változót, ami a felhasználó egyenlegét mutatja meg.
  + listFarmersMenu(): ha ezt választja a felhasználó, akkor egy for ciklussal, ami annyiszor fog lefutni, ahány elemű a farmers Arraylist kiíratja a farmerek neveit, árusított gyümölcseit, napi kívánt kereseteit.
  + listFruitsMenu(): ha ezt választja a felhasználó, akkor egy for ciklussal, ami annyiszor fog lefutni, ahány elemű a fruits Arraylist kiíratja a gyümölcsök neveit, romlásait, eladási árait.
  + listReceiptsMenu(): ha ezt választja a felhasználó, akkor egy for ciklussal, ami annyiszor fog lefutni, ahány elemű a receipts Arraylist kiíratja a múltbéli tranzakciók profitjait, tranzakció számát (azt mutatja meg, hogy hány darab gyümölcs szerepelt a tranzakcióban), tranzakció előtti és utáni előleget.
  + updateFarmerMenu(): ha ezt választja a felhasználó, akkor be kell írnia egy nevet, kikeresi a farmers listából a név indexét és megkéri a felhasználót hogy 1-3 között egy számot is adjon ami megadja egy switch case-szel, hogy melyik tulajdonságát akarja változtatni, majd a név indexére beilleszti az új tulajdonságot a Farmers osztály set függvényével
  + deleteFarmerMenu(): ha ezt választja a felhasználó, akkor bekér egy nevet, farmers listából kiválasztja a bekért név indexét majd az Array osztály .remove(by index) függvényével törli az azon az indexen szereplő farmert
  + addFarmerMenu(): ha ezt választja a felhasználó, akkor az Array osztály .add() függvényével, amiben használjuk a new Farmers() paramérteres konstruktort készíti el az új farmer, 3 lépésben: kér egy nevet a felhasználótól, egy árult gyümölcsöt és egy kívánt napi jövedelmet, majd ezekkel az adatokkal alkalmazza a konstruktor, az azzal készített Farmers típusú objektumot pedig hozzáadja a farmers Arraylisthez
  + tradeMenu(): ha ezt választja a felhasználó, akkor a program automatikusan elvégzi az elérhető kereskedések közül azokat, amelyek profitja nagyobb mint 0. Egy for ciklussal, amely annyiszor fut le ahány farmer van a farmers listában majd berakja a marketProfit() függvénnyel kapott profitjukat a profits listába illetve a nevüket a farmerName listába. Ezeket utána egy for ciklussal ami annyiszor fut le, ahány eleme van a profits listának, csökkenő sorrendbe rendezi. Majd ebből a listából elkezd balról jobbra vásárolni. Tehát

vásárol a profits lista első helyén szereplő profiton egy darab gyümölsöt a lista első helyén szereplő famrertől. Ha a farmer gyümölcse elfogyott, akkor a lista következő indexén szereplő farmer gyümölcseit veszi. Ezt egy while ciklusban ismételgeti, amíg legalább egy dolog ebből a háromból be nem következik: elfogytak azok a farmerek, akiknek pozitív profittal van gyümölcsük, vagy elfogyott a felhasználó egyenlege, vagy elfogynak a farmerek listából. Ezután visszaad négy adatot: a tranzakció során keletkezett összes profitot, a vásárolt gyümölcsök darabszámát, a tranzakció előtti egyenleget és a tranzakció utáni egyenleget. Ebből a négy adatból csinál egy receipts lista elemet, ami Receipts típusú, így a Receipts osztály konstruktorát alkalmazza.

* + XmlToFarmer(): ezt nem tudja a menüből meghívni közvetlenül a felhasználó. A farmers.xml fájlból beolvassa a farmers Arraylistbe a farmereket. Mivel a farmers.xml fájl úgy keletkezik, hogy a kilépésnél a saveFarmersToXml hozza létre, ezért ez a két függvény lehetővé teszi, hogyha futtatjuk a programot, majd bezárjuk, akkor a következő futtatásnál az első futtatás adatai ne vesszenek el. Tehát Mentés. Ez úgy működik, hogy egy tagListen megy végig egy for each ciklus és nodeListbe rendezi az elementeket, majd típus szerint kiválogatja őket és Implementálja a farmers lista Farmers osztály konstruktorához.
  + XmlToReceipt(): ezt nem tudja a menüből meghívni közvetlenül a felhasználó. A receipts.xml fájlból beolvassa a receipts Arraylistbe a vásárlási előzményeket. Mivel a receipts.xml fájl úgy keletkezik, hogy a kilépésnél a saveReceiptsToXml hozza létre, ezért ez a két függvény lehetővé teszi, hogyha futtatjuk a programot, majd bezárjuk, akkor a következő futtatásnál az első futtatás adatai ne vesszenek el. Tehát Mentés. Ez úgy működik, hogy egy tagListen megy végig egy for each ciklus és nodeListbe rendezi az elementeket, majd típus szerint kiválogatja őket és Implementálja a receipts lista Receipts osztály konstruktorához.

Hibakezelés

Felmerül a hibák lehetősége. Például, ha megkérjük a programot, hogy írja ki a farmers listát, de a farmers listához még nem adtunk egy farmert sem, akkor excepciont kapunk, hiszem a farmers akkor még null, amit nem írathatunk ki. De erre az esetre és az összes ehhez hasonlóra rendelkezik hibakezeléssel a program. Vagy try catch-el, például kér egy Integert, a felhasználó egy Stringet ad meg, akkor a catch kiírja, hogy “legközelebb egész számot adjon meg” majd újra meghívja az adott menüpontot, amíg a felhasználó helyes adatot nem visz be, vagy csak visszadobja főmenübe. De nem mindenhol alkalmaztam try catch-et, volt ahol például tudtam, hogy az első alkalmazásnál outOfBoundExceptiont kapnék, úgyhogy beletettem egy if-be, ami megvizsgálta hogy most fut e először le az adott dolog és ha igen, akkor az alapértéket használja, ha nem akkor az előző futásból szedje ki a szükséges adatot.