HÁZI FELADAT

Programozás alapjai 2.

Feladatválasztás/feladatspecifikáció

Gutási Ádám
XW53QZ

2022. 05. 16.

TARTALOM

1. Feladat

Tervezze meg a **Meseországi Villamos Művek (MVM)** nyilvántartási rendszerének egyszerűsített objektummodelljét, majd valósítsa azt meg! A rendszerrel minimum a következő műveleteket kívánjuk elvégezni:

- ügyfél adatinak felvétele
- szolgáltatási szerződés kötése
- szolgáltatási díj előírása (számlázás) (becsült átlagos fogyasztás)
- szolgáltatási díj befizetése
- egyenleg lekérdezése
- fogyasztás bejelentése

A rendszer lehet bővebb funkcionalitású, ezért nagyon fontos, hogy jól határozza meg az objektumokat és azok felelősségét.
Demonstrálja a működést külön modulként fordított tesztprogrammal!
A megoldáshoz **ne** használjon STL tárolót!

2. Feladatspecifikáció

A feladat egy áramszolgáltató nyilvántartási rendszerének egyszerűsített objektummodelljének megvalósítása. A feladat nem specifikálja az adatok tárolásának módját, így a futás alatt dinamikus tömbök (Array) használata mellett döntöttem. A teszteléshez használt minta adatokat szövegfájlokba rendezve, az összetartozó adatokat tabulátorral
tagolva fogom tárolni. A következő adatok lesznek elérhetőek külső fájlokból::

1. Ügyféladatok[1] (Vevőazonosító[2],*Vezetéknév, Keresztnév, Születési dátum, Lakcím, Telefonszám, e-mail cím, magánszemély/vállalkozás, adószám, legutolsó bejelentett óraállás, egyenleg, fázisok száma, főbiztosíték áramerőssége*)[3]
2. Fogyasztás bejelentések (induló és zaróállás, vevőazonosító)
3. Előre kiírt számlák (ha az ügyfél elkésik a bejelentéssel, vagy az egyenlege alapján

   levonás után nincs mit fizetnie).

4. Teljesített számlák
5. Villamos energia tarifák
6. naplófájl (nem tartozik szorosan a teszteléshez)

A program indulásakor a parancssorban egy menü jelenik meg, ahol a megfelelő szám beírásával ki lehet választani, milyen műveletet szeretnénk végrehajtani. Az adott művelet végrehajtása után visszatérünk ugyanebbe a menübe, egészen addig, amíg a kilépés opciót nem választjuk.
A teszt megírásához a **gtest_lite** könyvtárat fogom használni, az esetleges memóriaszivárgások felderítésére pedig a **memtrace** könyvtárat.

[1] Az ügyféladatok egyértelműen azonosítják a szerződés adatait is, így nincs szükség azok külön tárolására.
[2] Egy ügyfél létezhet a rendszerben többször is, több azonosítóval (magánszemélyként/vállalkozásként, akár azokban többször is). Egy vevőazonosítóhoz egy mérőóra tartozik.
[3] Két ügyféltípust fog megkülönböztetni a rendszer: magánszemély és vállalkozás. A *vállalkozás* ügyféltípusnál értelemszerűen a megadott személyes adatok a cég kapcsolattartójának adatai. A két kategóriára külön árszabás lesz érvényben.

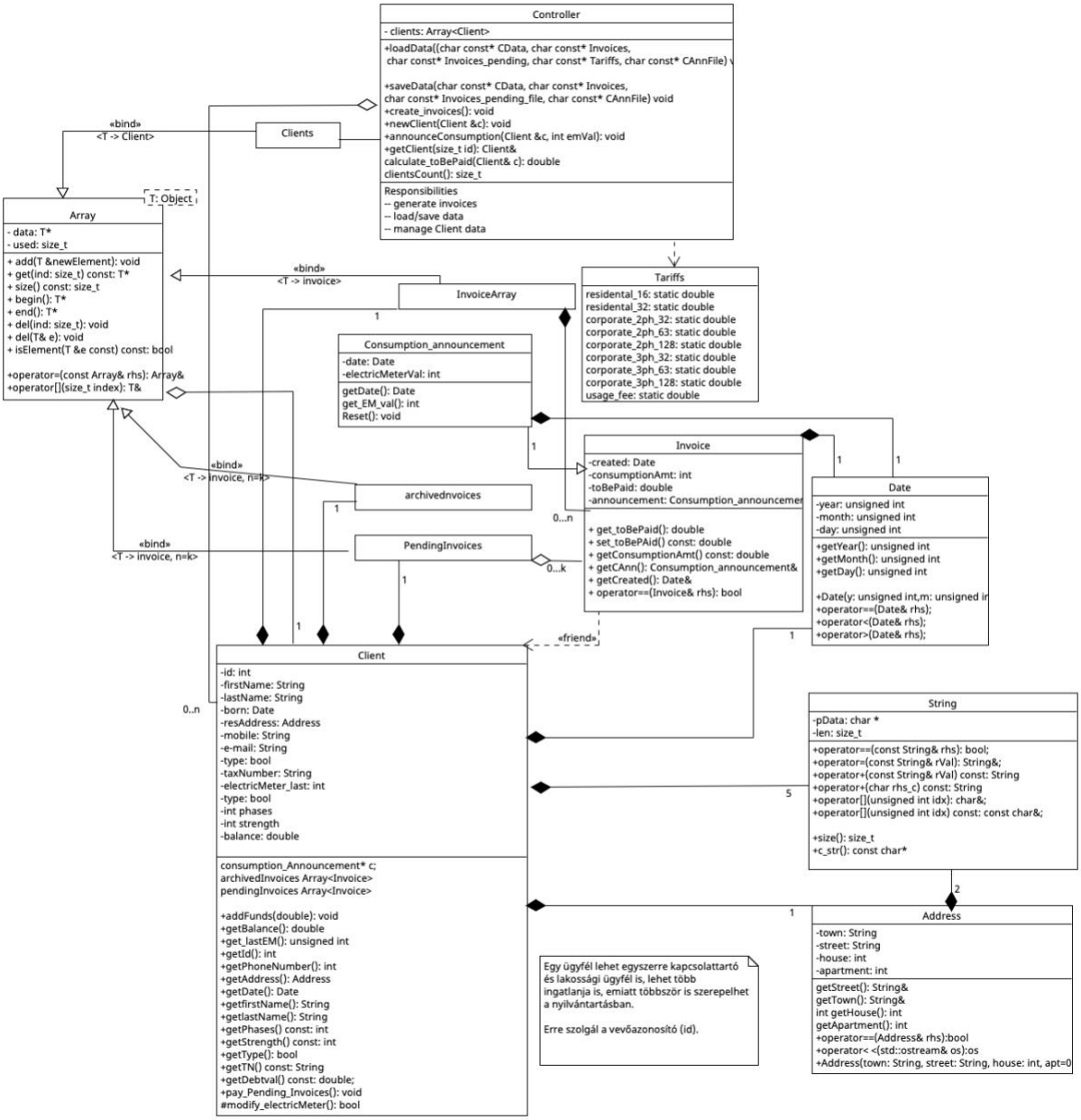
A fizetendő tartalmazza a fogyasztott áram mennyiségéből, az árszabás alapján számított díjat, a rendszerhasználati díjat, és az ÁFA mértékét (ügyféltípustól változó). Ha az ügyfél túlfizeti számláját (tehát egyenlege a befizetés után pozitív), akkor a következő számlázási időszakban (következő hónapban) a kiállított számla tartalmazza az elszámolás értékét (tehát túlfizetett számlánál levonódik a maximális egyenleg az ügyfél egyenlegéből.)

A saját áramot termelő ügyfelek esetén feltételezzük, hogy a bejelentett mennyiségből már ki van vonva az általuk termelt árammennyiség. Feltételezzük továbbá, hogy ezen ügyfelek mérőórái nem mozdulnak el termelés hatására.

Ha nem érkezett az adott hónapra fogyasztás bejelentés, és sikeres teljesítés, akkor előre kiír számlát a program, az előző teljesített számlák átlagai alapján meghatározott mennyiséggel. Ezeket az eseményeket egy naplófájlban lehet követni, tesztelés és debug céljából. A naplózás a CPORTA flag használatával kikapcsolható.

Az ügyfél nem tud egy-egy számlát befizetni. A befizetés menete az, hogy befizetni a pénzt az MVM-nek, ahol ügyfelenként tárolva van az egyenleg, majd amennyiben fedezi a számlát a bennlevő összeg, a rendszer levonja a fizetendőt az ügyfél egyenlegéről, a megmaradt összeg pedig túlfizetésnek minősül. Ha az ügyfélnek több számlája is befizetésre vár, akkor időrendi sorrendben, a legrégebbitől haladva próbálja teljesíteni azt a rendszer.

3. Osztálydiagram



A **Controller** osztály fogja össze a teljes struktúrát. A menün keresztül leadott parancsainkat a Controller függvényei teljesítik. Az adatokat betöltés után Array osztályok tárolják.

Egy Ügyfélhez egyidőben csak egy fogyasztási bejelentés tartozhat. A számlák generálása után törlődik, fogyasztás bejelentésének hatására létrejön.

4. Fontosabb algoritmusok

4.1. generate_invoices()

ez a függvény minden hónap végén lefut, és a beérkezett bejelentések/átlagok alapján kiírja a számlákat.

```
i:=0; l:=0;
ciklus amíg i!=(Ügyfelek_száma):

ha (van fogyasztási bejelentés): sz:=számlázás(k);

fizetendő_számlák_hozzáad(sz);

egyébként:

j:=átlag_óraállás(Ügyfél); k:=fogyasztási_bejelentés_létrehoz(j,i);
sz:=számlázás(k); fizetendő_számlák_hozzáad(sz);

ha (fizetendő számlák mennyisége>1): //van olyan régebbi számla, ami még
befizetésre vár.

ciklus amíg egyenleg>(fizetendő_számlák[l].fizetendő()) vagy
l<fizetendő_számlák[l].hossz()):

teljesít_számla(fizetendő_számlák[l]);

számla_archivál(fizetendő_számlák[l]);
különben: // nincs tartozás, megpróbáljuk befizetni a most kiírt számlát.

ha(teljesít_számla(sz)): számla_archivál(sz);

i+=1;

ciklus vége
eljárás vége
```

4.2. avgConsumption(Client &c): int

`átlag_óraállás(Ügyfél)`– átlagos fogyasztást számít az ügyfélnek, ha nem jelentette be az óraállását.

```
i:=0; s:=0;
ciklus amíg i<(archivált_számlák.hossz())

     s+=archivált_számlák[i].fogyasztás();

ciklus vége
RETURN egészrész(s/i);

eljárás vége
```

5.