Beléptető rendszer készítése előre meg nem határozott objektumra Java Kártya (Java Card) ill. Java Gyűrű (Java Ring) felhasználásával.

Készítette: Szatmáry Botond
ELTE TTK, Programtervező Matematikus szak, IV. évf.

Köszönet Kincses Zoltánnak a segítségért

Tartalomjegyzék

1. Bevezetés **2**

2. Java kártya (Java Card) **2**
2.1. Felépítése 3
2.2. Inicializálás és megszemélyesítés 3
2.2.1. Kommunikáció 4
2.3. Biztonság 5

3. A konkrét feladat **6**
3.1. A rendszer általános leírása 6
3.1.1. Központi gép 6
3.1.2. Terminálok, gyűrűk 6
3.2. A rendszer működése 11
3.2.1. Új objektum tervezése, megvalósítása 11
3.2.2. Már meglévő épület/objektum módosítása 13
3.2.3. Új személy felvétele 13
3.2.4. Létező rendszer üzemeltetése, napi használata 14

4. Összefoglalás **16**

Forráslista és Irodalomjegyzék **18**

1. Bevezetés

A feladat a következő: beléptető rendszer tervezésére, működtetésére alkalmas rendszert kell készíteni. Tervezés alatt értendő, hogy hova kell a kapukat tenni, ezek milyen jogosultságokkal rendelkezzenek, milyen korlátokkal működjenek. Működtetés pedig a rendszer használatát jelenti mind a felhasználók mind az üzemeltetők részéről. A felhasználó nyilván akkor elégedett, ha könnyen, várakozás nélkül járkálhat mindenhova, használhat mindent az objektumon belül, amihez jogosultsága van, és ugyanakkor biztonságot is nyújt számára. Az üzemeltető számára akkor jó a rendszer, ha könnyen ellenőrizhető, karbantartható, rugalmas, megfelelően biztonságos.

Az utóbbi időben (egy-két évtizede) a hagyományos módszereken kívül a beléptető rendszerek tervezésénél, alkalmazásánál egyre inkább használnak mágnes ill. chipkártyás megoldást a régi helyett, ahol egy „fontos" ember (portás) jelentős méretű kulcscsomóval mászkált, hogy mindenhova bebocsátást szerezzen magának. Kártyás megoldással ez a probléma kiküszöbölhető, de nem ez a fő előnye, hanem, hogy biztonságosabb ez a megoldás, hiszen össze lehet kötni PIN (Personal Identification Number) kód, ujjlenyomat ellenőrzéssel. Én Java Kártya szabványnak megfelelő Java Gyűrűt használtam. A Java Kártya egy speciális chipkártya (intelligens kártya, smart card), a gyűrű pedig gyakorlatilag ugyanezt tudja, csak formája különbözik. A specialitás abból adódik, hogy itt majdnem mindent a kártya/gyűrű tud: rendelkezik memóriával, egy korlátozott virtuális géppel, így rá lehet tölteni alkalmazásokat, melyeket később használni lehet és a terminál elég, ha csak egy 'buta' olvasó. A gyűrű a memória tartalmát a gyártó cég (Dallas Semiconductor Corp.) szerint 10 évig is megőrzi akár szélsőséges körülmények között is, egy lithium elem segítségével.

Ösztönző volt számomra, hogy most épül/épült az egyetem új épülete, ahol esetleg fel lehetne használni az elkészült programot/alkalmazást akár kis módosítással az új diákigazolvány nyújtotta(?) lehetőségeket is kihasználva.

2. Java kártya (Java Card)

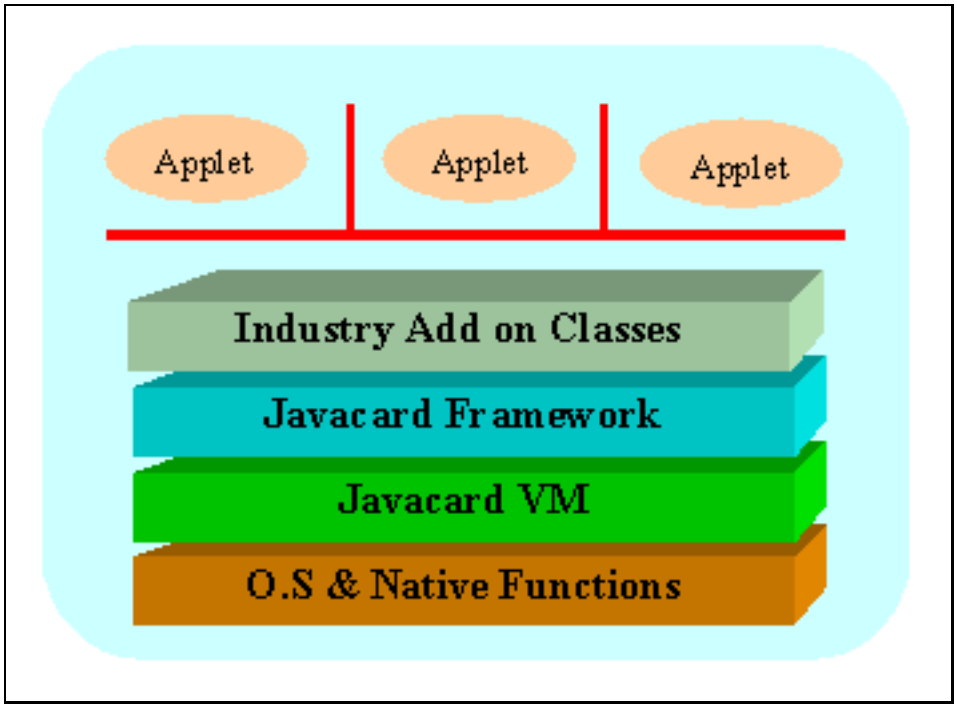
A számítástechnika fejlődése során egyre inkább fellép a platformfüggetlenség igénye. Ennek az igénynek a kielégítésére született a Java nyelv. Majd később megjelentek a Java nyelvet használó intelligens kártyák. Ezek a kártyák Java programokat képesek futtatni jelenleg a Java Card 2.0 specifikáció alapján. Ennek a nagy előnye abban rejlik, hogy nem kell a programokat a kártyán tárolni (a kártyára „töltődnek le"), a felszabadult memóriaterületet más adatok tárolására lehet felhasználni, és bármilyen változás esetén nem

igényli az egész kártya cseréjét, hanem csak a központi alkalmazás cseréjét.

2.1. Felépítése

A Java kártya minimálisan 16 Kbyte csak olvasható memóriát (ROM), 8 Kbyte EEPROM-ot, és 256 byte RAM-ot tartalmaz.

A kártya architektúrája a következő:



Amint az ábrán látható, a Java Card VM (virtual machine) az integrált áramkör (IC) és az operációs rendszer felett helyezkedik el. A JVM réteg elrejti a rendszer specifikus részeket és egy általános felületet nyújt. A Java Card Framework egy programozási felületet (API, Application Programming Interface) kínál, mely Java Card alkalmazások fejlesztését teszi lehetővé. Az Industry Add on Classes réteg segítségével saját funkciókkal bővíthetjük a kártya szolgáltatásait (például: biztonsági funkciók). A Java Kártya alkalmazásokat „cardlet”-eknek nevezzük. Több ilyen appletet is rátölthetünk a kártyára. Minden egyes cardlet egy egyértelmű azonosítóval rendelkezik, AID (applcation identifier, ISO 7816-5)

2.2. Inicializálás és megszemélyesítés

Minden kártyának mielőtt használni lehetne át kell esnie egy inicializáló, és egy megszemélyesítő eljáráson. Az inicializáló rész a kártya nem törlődő me-

móriájába általános adatok felvitelét jelenti. Ilyen adatok például a gyártó neve, a kártya sorszáma stb. A megszemélyesítés történhet fizikailag és elektronikusan. Fizikai megszemélyesítés esetén a kártya felületébe kerülnek fel a személyes adatok (például lézer gravírozással), elektronikus módon pedig a kártya állandó (nem törlődő) memóriájába kerülnek be az adatok.

2.2.1. Kommunikáció

A kártyák a külvilággal - hasonlóan két számítógép adatcseréjéhez - saját adat csomagokkal kommunikál (APDU, Application Protocol Data Units). Az APDU vagy egy parancsot, vagy egy választ tartalmaz. A kártyák világában a „master-slave" modell használatos, a kártya mindig a passzív szerepet játssza. Pontosabban, a kártya mindig egy parancsra vár, majd végrehajtja azt, és válaszol. Az APDU struktúrája az ISO 7816-4 szabványban rögzített:

APDU parancs						
Fejléc				Törzs		
CLA	INS	P1	P2	Lc	Adatmező	Le

A fejléc 4 mezőt tartalmaz: objektum CLA, utasítás INS, paraméterek P1, P2. Mindegyik mező 1 byte hosszú.

– CLA: Class byte. A legtöbb kártyánál egy alkalmazás azonosítására szolgál.

– INS: Instruction byte. Az utasítás kódja.

– P1, P2: Parameter bytes. A paraméterek.

A törzs részben az Lc az Adatmezőben található byte-ok számát tartalmazza, Le pedig a parancsot követő válasz Adatmezőjében található byte-ok maximális számát adja meg:

APDU válasz		
Törzs	Adatmező	
Követő adatok	SW1	SW2

Az SW1, SW2 státusz byte-ok. A feldolgozott parancs végrehajtásáról adnak információkat (például: sikeres volt vagy sem).

A kártyák a külvilággal - hasonlóan két számítógép adatcseréjéhez - saját adat csomagokkal kommunikál (APDU, Application Protocol Data Units). Az APDU vagy egy parancsot, vagy egy választ tartalmaz. A kártyák világában a „master-slave" modell használatos, a kártya mindig a passzív szerepet játssza. Pontosabban, a kártya mindig egy parancsra vár, majd végrehajtja azt, és válaszol. Az APDU struktúrája az ISO 7816-4 szabványban rögzített:

APDU parancs						
Fejléc				Törzs		
CLA	INS	P1	P2	Lc	Adatmező	Le

A fejléc 4 mezőt tartalmaz: objektum CLA, utasítás INS, paraméterek P1, P2. Mindegyik mező 1 byte hosszú.

– CLA: Class byte. A legtöbb kártyánál egy alkalmazás azonosítására szolgál.

– INS: Instruction byte. Az utasítás kódja.

– P1, P2: Parameter bytes. A paraméterek.

A törzs részben az Lc az Adatmezőben található byte-ok számát tartalmazza, Le pedig a parancsot követő válasz Adatmezőjében található byte-ok maximális számát adja meg:

APDU válasz		
Törzs	Adatmező	
Követő adatok	SW1	SW2

Az SW1, SW2 státusz byte-ok. A feldolgozott parancs végrehajtásáról adnak információkat (például: sikeres volt vagy sem).

A következő programrészlet a TerminaltFogad.java file-ból van, mutatja, hogy a kódban a kommunikáció (pl. olvasás) hogyan jelenik meg:

```
byte[] buffer = new byte[0];
CommandAPDU businessCardAPDU = new
            CommandAPDU( (byte)T_CLA, (byte) T_READ, (byte)0,
 (byte)0, buffer, (byte)0 );

ResponseAPDU response = channel.sendAPDU(businessCardAPDU);
validateResponseData(response);
byteArrayToFields(response.getDataBytes());
```

Ahol T_CLA, T_READ konstansok, melyek meghatározzák, hogy melyik gyűrűn lévő osztálytól, milyen műveletet végrehajtását kéri.

2.3. Biztonság

A kártya specifikációja tartalmazza a manapság használt kódolási eljárásokhoz (DES, 3DES, RSA) szükséges osztályokat, metódusokat. A tervezők a cardletek futásánál is ügyeltek a biztonságra. A cardletek objektumokat hozhatnak létre, melyek adatokkal dolgoznak. Minden objektumnak a tulajdonosa az őt létrehozó cardlet. Egy cardlet csak akkor férhet hozzá egy objektumhoz, ha az ő tulajdona, vagy pedig meg van osztva. Egy cardlet a saját objektumait megoszthatja egy másik cardlettel, vagy pedig az összes cardlettel.

3. A konkrét feladat

3.1. A rendszer általános leírása

Adott egy központi gép (esetleg megoldható úgy is, hogy több központi gép legyen, így az elsődleges gép meghibásodása esetén egy másik gép vehetné át a szerepét) és több, akár korlátlan számú terminál, amelyek kapuknál lesznek elhelyezve, ezeken keresztül lehet az objektumba bejutni illetve azon belül közlekedni.

3.1.1. Központi gép

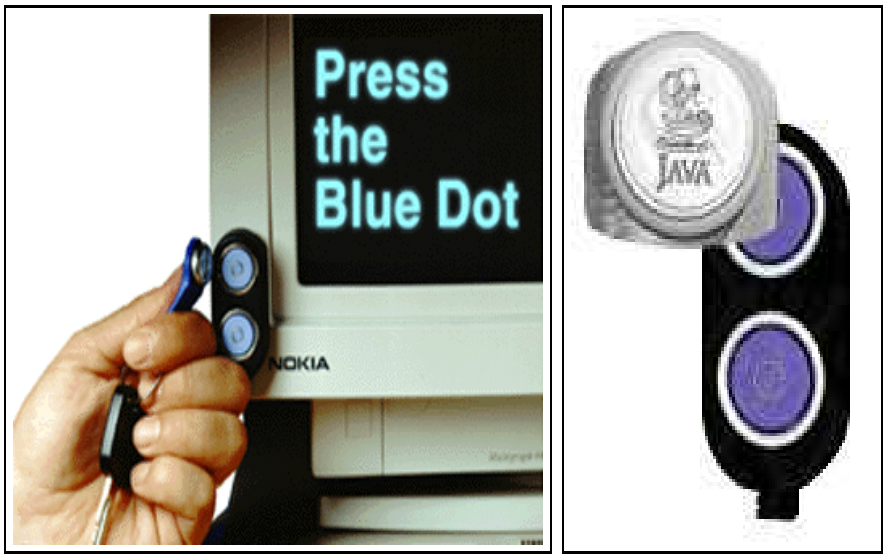
A központi gép az lehet bármilyen PC, vagy számítógép, amely Java virtuális gép futtatására alkalmas, és lehet hozzá több (pontosan annyit ahány terminál van) gyűrűolvasót kapcsolni. Gyűrű olvasó az vagy párhuzamos, vagy soros porton tud a számítógéphez kapcsolódni. Ehhez megfelelő driver is kell. A gyűrűolvasó ill. gyűrű gyártója biztosít egy fejlesztőkörnyezetet, és ebben benne van három operációs rendszerhez (Microsoft Windows 95, Windows NT, Linux, Solaris) is a megfelelő vezérlő. A terminálok (gyűrűolvasók) ehhez a központi számítógéphez kapcsolódnak és továbbítják felé, illetve kérnek tőle információt. Ezen a gépen, vagy bármely más Java programok futtatására alkalmas gépen meg kell tervezni az épületet/objektumot aminek a felügyeletét végzi majd ez a gép. Az elmentett tervet erre a gépre kell másolni, mert ezt felhasználva tud működni a felügyelő program. A felügyelő program az üzemeltetést végzi, ehhez fordulnak kéréssel a terminálok, ez ismeri az objektum szerkezetét, ez végez naplózást.

Az elmentett objektum irányított gráf segítségével van tárolva. Minden egyes csúcsnak egy terminál felel meg. Ha két csúcs között van él, akkor az azt jelenti, hogy az egyik terminálótól a másikig az objektumon belül létezik út (ez az út fizikai utat jelent és nem egy gráfbeli utat). Ez a program tudja, hogy melyik terminál milyen tulajdonságokkal rendelkezik. A terminálok maguk „buták", minden információ a gyűrűn illetve ebben a kezelő programban van. A gyűrű tárol minden szükséges információt a tulajdonosáról, a kezelő program pedig a teljes gráfot tartalmazza. Így a kettő együtt (központi gép és gyűrű) a beolvasó szerepet betöltő terminál segítségével egyértelműen eldöntheti, hogy adott belépési szándékkal érkező személy átmehet a kapun vagy sem.

3.1.2. Terminálok, gyűrűk

A terminálok jelen esetben a gyűrű író/olvasó egységek. Eredeti nevük: Blue Dot Receptor. Szintén a Dallas Semiconductor Corp. terméke.

A Blue Dot Receptor teszi lehetővé a kapcsolatot az iButton (gyűrű) számára. Ennek a kábele csatlakozhat soros illetve párhuzamos porton keresztül is. A Dallas Semiconductor Corp. mindkét típushoz biztosítja a megfelelő vezérlőt Linux, Solaris, Win95 és Windows NT operációs rendszerekhez. A kapcsolat a gyűrű és a host gép között aktivizálódik, ha a gyűrűt (iButton) hozzáérintjük az olvasóhoz



Blue Dot Receptor. *Ezen keresztül tud a gyűrű (iButton) kommunikálni a gazda géppel.*

(Blue Dot-hoz). Ha akarjuk, rögzíthetjük is rajta, ez kényelmesebb ha hosszabb műveletet végzünk.

Az iButton (gyűrű) tartalmaz egy 16 mm^2-es processzort. Mindez acélkeretben van, amely ellenállóvá teszi, így az kibír ütést, vizet, áramot, más szélsőséges körülményt is, ezzel lehetővé téve a napi használatát. Minden gyűrű a gyártó (Dallas Semiconductor Corp.) által garantált egyedi 8 byteos regisztrációs számmal rendelkezik. Van benne NV SRAM gyors memória amiben szöveg, kép, alkalmazások, bármi tárolható. Ezt a tartalmát akár 10 évig is megőrzi. Módosítani a Blue Dot Receptoron keresztül lehet. Egyszerre több alkalmazás is rátölthető, ezeket az alkalmazásokat egyedi azonosítójuk segítségével érhetjük el. A gyűrűkön van egy Java Virtuális Gép, ami megfelel a JavaCard 2.0 specifikációnak. Ebből kiemelve:

- Igazi 32 bites Java egészeket kezel
- Automatikus szemét gyűjtő (garbage collector) van a kedvező memóriahasználat érdekében
- Applikációkat lehet törölni, rátölteni biztonságos módon
- Egyedi 64 bites regisztrációs szám
- Titkosítási eljárásokat (RSA, DES) támogat
- Véletlenszám generátor

A gyűrű számítási képességei:

iButton – Java Gyűrű. *Acél keretbe foglalt gyűrű formájú JavaCard 2.0 specifikációnak megfelelő 16 mm^2-es chipet tartalmaz.*

- 0.6 mikronos CMOS (3V-tól 5V-ig)
- 800000 tranzisztor növeli a számítási képességeit
- gyors processzor, mely saját magának állítja az órajelét, megelőzve a programból való illetéktelen hozzáférést
- 32 kbyte ROM a Java Virtuális Gépnek és az operációs rendszernek
- matematikai számítások gyorsítása, 1024 bites RSA visszafejtést kevesebb mint 1 másodperc alatt

Amint már említve volt, a központi gépen a felügyelő programban egy gráf ír le egy objektumot. A gráf minden egyes csúcsa egyértelműen megfeleltethető egy terminálnak, az élek pedig az objektumon belüli (fizikai) utaknak feleltethetők meg. Egy csúcsot és így szükségszerűen terminált is egyértelműen meghatározza az azonosítója, tehát amikor egy terminál, kéréssel fordul a géphez, akkor tudni lehet a helyét a gráfban.

Minden egyes csúcs leírja:

- Milyen élek vezetnek ki belőle, azaz adott terminálból közvetlenül melyik másikig lehet eljutni.
- A terminál tulajdonságait:
  - Azonosítója. Ez alapján lehet a gráfban rákeresni. Nem szabad két különböző terminál legyen megegyező azonosítóval.
  - Neve. Ez nem kell egyedi legyen.

- Mely csoporthoz tartozó emberek mehetnek át rajta.
- Hol helyezkedik el az objektumon belül: x, y, z koordináta.
- Időintervallum: normál esetben mikor engedélyezi a rajta átmenő forgalmat.
- Történetvektor: megadja, hogy kik és mikor haladtak át rajta. Ennek a vektornak a méretére érdemes valamilyen korlátot megadni a túlzott memóriafogyasztás elkerülése érdekében.

Tehát a terminálok között hierarchikus kapcsolat van. Az egyértelműség kedvéért mindig létezik egy gyökérelem ahonnan bármelyik csúcsot a gráfban el lehet érni. Ez a gyökér automatikusan létrejön, amikor új objektum tervezésébe kezd a felhasználó.

Ez az egész hierarchikus ábrázolás azért érdekes, mert ennek segítségével a következő plusz biztonsági elemet lehet a rendszerbe beépíteni: Annak eldöntése, hogy egy ember átmehet-e egy kapun az nem csak az alapján dől el, hogy megfelelő-e a csoportja, hanem az is számít, hogy mi volt az előző terminál ahol járt. Ez azért fontos, mert így elkerülhető az, hogy az emberek egy forgalmasabb kapunál tömegesen menjenek át úgy, hogy csak az első ember használja a gyűrűjét, hiszen ha így tesznek, akkor nem engedi őket tovább a következő kapun, mert illegális közlekedésre történő kísérletet észlel. Ez még egy típusú betörés ellen jó: Valaki ellop egy kártyát majd illegálisan bejut az épületbe (portást kijátszva, terminált nem használva), de mégsem tud közlekedni, hiszen a rendszer érzékeli, hogy eddig nem használt terminált, egyértelműen kiderül, hogy az adott illető nem szabályosan került az épületbe.

A rendszert (terminálokat) szükség esetén el lehet látni még további biztonsági elemekkel, mert ezek nélkül a gyűrű ellopása túl sok jogot megadna a tolvajnak. A terminálokba be lehet építeni PIN kód ellenőrzést, vagy ami még megbízhatóbb, hogy ha ujjlenyomatot ellenőriz. Mivel gyűrűt nem lehet bevonni, ezért többszöri illegális PIN kód beütése esetén a rendszernek riasztania kell. Hasonlóan kell eljárni, ha (ujjlenyomat ellenőrzés esetén) az ujjlenyomat nem egyezik. Ha ujjlenyomat vizsgálatot építünk a rendszerbe, akkor a gyűrűn tárolt, a fent említett tulajdonságok mellé felkerül más is: a gyűrű kiadásakor a leendő tulajdonos ujjáról többször mintát kell venni, majd ezen összesített adatokból, egy azt az ujjat leíró matematikai formulát lehet nyerni. Általában 25-30 pontot tárolnak egy ujjról. Belépéskor, ellenőrzéskor is mintát vesznek, majd összehasonlítják a tároltakkal. Elfogadott gyakorlat, hogy már 12 pont egyezése esetén egyértelműnek tekintik a tulajdonost, tehát 12 pont egyezése esetén a rendszer is elfogadja a személyazonosságot.

A terminál–gyűrű páros a program során két funkciót lát el:

1. Új személyt vesz fel az adminisztrátor az eddigiek mellé. Ilyenkor az új ember kap egy egyedi gyűrűt, amelyre rá kell tölteni egy megfelelő applet. A

DermoPrint Kft. terméke. *Nemzetközileg elismert, híres magyarországi cég ujjlenyomat felismerő rendszere.*

központi gép, amikor a gyűrűvel kommunikál, akkor sokszor ennek az applettnek a szolgáltatásait veszi igénybe. Ezek után az adminisztrátor tárolja rajta az új ember adatait.

2. Egy már felvett személy jelentkezik valamelyik terminálhoz, beilleszti a gyűrűt az olvasóba, az ezt az eseményt érzékeli, leolvassa az adatokat a gyűrűről, majd eldönti, hogy az átlépést engedélyezi vagy sem. Mindeközben naplózás történik/het mindkét helyen: a gyűrűn tárolódik, hogy tulajdonosa milyen útvonalon közlekedik, a központi gépben pedig terminálok szerint lebontva mentődik el ugyanaz az információ.

3.2. A rendszer működése

3.2.1. Új objektum tervezése, megvalósítása

Mielőtt a tervező hozzákezdene egy új objektum tervezésébe alaposan át kell gondolnia az egészet. Ki kell találni a biztonsági és hatékonysági szempontokat is figyelembe véve, hogy hova és milyen terminálokat helyezzen el. Nem szabad túl sűrűn elhelyezni azokat, mert akkor az épület használóinak nagyon megnehezítené a helyzetét, és arra kényszerítené, hogy kijátsszák a rendszert, így csökkenne annak a biztonsága, és nagyon lassú lenne, forgalmasabb időszakban tömegjelenetek lennének egy-egy kapunál, ajtónál. Tervezés folyamatába tartozik, hogy kitaláljuk mindegyik terminál tulajdonságát, milyen csoportokat enged magán keresztül, milyen más terminálok közelíthetőek meg közvetlenül. Ebben a fázisban kell kitalálni az időintervallumokat, hogy mikor milyen üzemmódban legyenek a terminálok:

- Általában a terminálóknak van egy rendes időszaka, ez normál működés esetén. pl.: a lovarda 8-tól 20 óráig diákokat, tanárokat stb. beenged, viszont, ha egy diák ezután akar bemenni vagy kijönni, akkor azt kivételes esetként kell kezelnie a rendszernek.

- Lehet a terminálóknak egy úgynevezett nyitott időszaka, amikor a hozzá tartozó ajtót/kaput abban az időintervallumban állandóan nyitva tartja. pl.: egy egyetemi előadó előadások idején nyugodtan lehet nyitva. Ha zárt körű előadás zajlik, akkor ellenőrizni kell a belépőket.

Létezik egy szerkesztő program (Takács Mihály készített), mely segítségével könnyedén, grafikus eszközökkel lehet egy új épület tervét elektronikus, később használható formára hozni, menteni. Ez a grafikus felület a gráf, csúcs, lista osztályok metódusait használja:

- `graf g = new graf(nev)` paranccsal lehet új gráfot létrehozni. `nev` lesz a gráf gyökerének azonosítója. Amikor új épület/objektum elkészítésébe fogunk ez kell legyen az első lépés. Ezután tudunk a gráfba új elemet azaz csúcsokat beilleszteni.

- `g.ujelem( neve, azonosito, csoport, kovetkezok, x, y, emelet, tol, ig )` metódussal lehet új csúcspontot a `g` gráfba illeszteni. A paraméterek a következők:
  - `String neve` a terminál/csúcs neve. pl.: „lovarda”
  - `String azonosito` a terminál/csúcs azonosítója.
  - `String csoport` a csoportja. Ez határozza meg, hogy milyen csoportokat enged át. pl.: „diák”
  - `lista kovetkezok` vagy `String kovetkezok` írják le, hogy mik a rákövetkezői ennek a csúcsnak. A `lista` a `Java.util.Vector` osztály leszármazottja. Ha lista típusú paraméterrel hívjuk meg, akkor a lista elemei azon csúcsok azonosítói, melyeket következőknek szánunk. Ha String típusú a `kovetkezok` paraméter, akkor ',' vesszővel elválasztva kell leírni a következő csúcsok azonosítóit.
  - `int x, int y, int emelet` paraméterek határozzák meg az x, y, z koordinátáit a csúcsnak az épületen/objektumon belül.
  - `String tol, String ig` paraméterek határozzák meg az időintervallumot, amikor a terminált normál működés mellett használni lehet. pl.: '8:00', '20:00' jelenti, hogy reggel 8-tól este 8-ig áll rendelkezésre. Fontos megjegyezni, hogy ha a `kovetkezok` között olyan azonosító szerepel, amelyhez nem tartozik csúcs (nincs még olyan azonosítójú csúcs), akkor létrejön egy ilyen azonosítójú csúcs, viszont semmilyen más paramétere nem lesz helyesen beállítva. Amikor majd erre az azonosítóra is meghívjuk az `ujelem` metódust, akkor állíthatjuk be azokat. Ha úgy hozunk létre csúcsot, hogy semmilyen másik nem mutat rá, akkor az elvész, tehát csak a gyökértől kifelé terjeszkedve lehet a gráfot felépíteni. Nem lehet erdőként a gráfot felépíteni, majd fává összehúzni.
- `g.torol( csucs p )` metódus a `g` gráfból kitörli a `p` csúcsot. Ez a hatékonyság érdekében csak egy törlés jelzőt állít át.
- `csucs g.Keres(csucs p, String azonosito)` metódus a `g` gráfban a `p` csúcs alatt keres `azonosito` azonosítójú csúcsot. Visszatérési értéke a csúcs amit megtalált, vagy `null`, ha nincs ilyen azonosítójú csúcs.
- `csucs g.Keres(String azonosito)` ugyanúgy működik mint az előző metódus, csak itt automatikusan `g` gráf gyökerétől lefelé keres.
- `csucs g.Keres(csucs p, int x, int y)` metódus a `g` gráfban a `p` csúcs alatt keres `x, y` koordináta alapján csúcsot. Visszatérési értéke a csúcs amit megtalált, vagy `null`, ha nincs ilyen azonosítójú csúcs.
- `csucs g.Keres(int x, int y)` ugyanúgy működik mint az előző metódus, csak itt automatikusan `g` gráf gyökerétől lefelé keres.

- `Java.util.Vector g.Helyes` metódus ellenőrzi, hogy a gráf csúcsai helyesen vannak-e feltöltve, azaz nincsen-e olyan csúcs amire csak mutat egy másik következője, de paraméterei nincsenek beállítva.

- `g.ment(String neve)` metódus kiírja a gráfot egy speciális formában `neve` nevű file-ba, ahonnan ha kell akkor be lehet újra tölteni.

3.2.2. Már meglévő épület/objektum módosítása

Módosításra akkor van szükség, ha a létrehozott, elmentett struktúra nem felel meg a követelményeinknek, vagy egy már jól működőt kell átépítés miatt bővíteni.

Előbb említettem, hogy a szerkesztés végén el lehet menteni a gráfot. Ha ezt módosítani akarjuk, akkor azt újra be kell tölteni, majd elvégezhető a javítás. Új csúcsot/terminált vehetünk fel, törölhetünk meglévőket, éleket hozhatunk létre, törölhetünk. A következő műveletek segítenek ebben:

- `graf g = new graf();` Amikor file-ból akarunk gráfot betölteni, akkor ezzel (üres) a konstruktorával kell példányosítani a `g` gráfot.

- `g.beolvas(file.nev)` metódus `file.nev` file-ból felépíti a gráfot garantáltan hibamentesen, ha ezt a `file.nev` file-t a `ment` eljárással hoztuk létre.

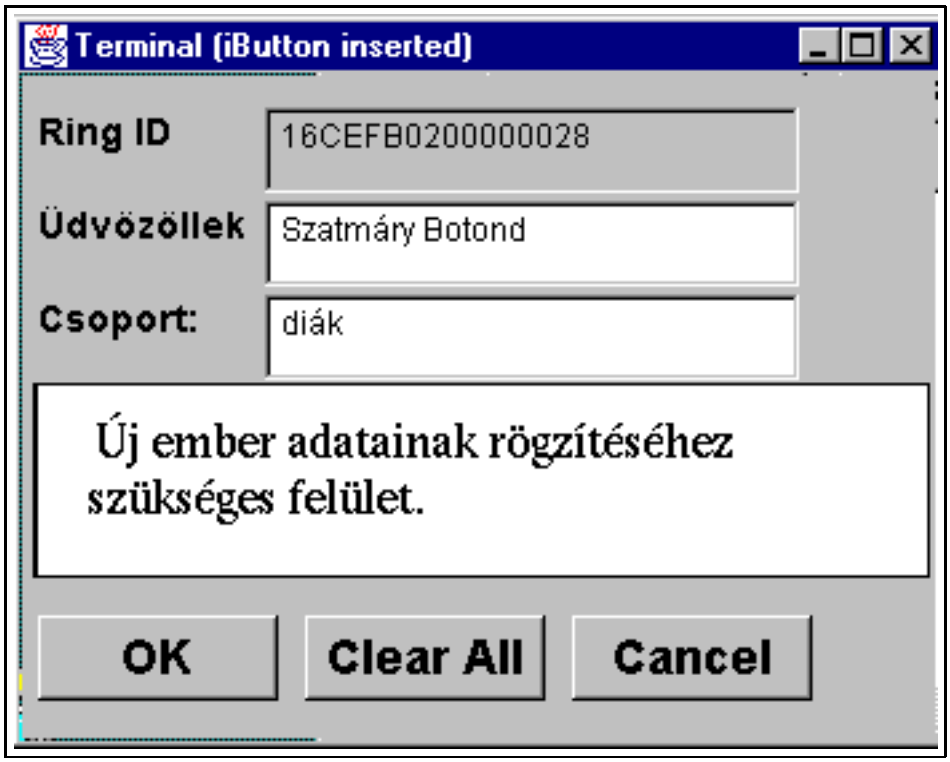
- `g.ujelem( neve, azonosito, csoport, kovetkezok, x, y, emelet, tol, ig )` metódussal nemcsak új csúcspontot lehet a `g` gráfba illeszteni, hanem már meglévőt módosítani. Paramétereket: lásd előző felsorolás. Az `azonosito` azonosítójú csúcs tulajdonságait módosítja az újakkal. A `neve, csoport, x, y, emelet, tol, ig` tulajdonságok felülíródnak. Ha a `g` gráf `azonosito` azonosítójú csúcsának következő mezője tartalmaz olyat ami a paraméterként átadott `kovetkezok` között nem szerepel, akkor törli azt az élet. Ha a paraméterként átadott `kovetkezok` között szerepel olyan azonosító amely a `g` gráf `azonosito` azonosítójú csúcsának következő mezője nem tartalmaz, akkor hozzáveszi az új élet.

- Értelemszerűen használhatjuk azokat a metódusokat, amely az előbbi fejezetben említettem (`Keres, Helyes, ment, torol`).

3.2.3. Új személy felvétele

Új személy felvétele az új gyűrű kiadását is jelenti. Mindezeket nem teheti meg bárki. A programok egy része (legalább a PIN kód amit használnak) nem publikus, így hacsak el nem lopják ezeket az adatokat, akkor nem tudják a gyűrű memóriáját értelmesen átírni.

Új személyt bármikor felvehetünk. Az új ember kap egy egyedi gyűrűt, arra rá kell tölteni azt az alkalmazást, amelyet a rendszer használ. Ez jelen esetben a `Terminal.class` file amit `Terminal.jib` file-ba kellett konvertálni a gyártó által szolgáltatott programmal. Ezután pedig rámásolható az új felhasználó neve és



Új ember adatait lehet tárolni. *Ebbe az ablakba kell beírni a megfelelő adatokat, majd rögzíteni. Ezek később egy egyszerű felhasználó számára nem módosíthatóak. A tulajdonosa sem tudja átírni.*

csoportja. Ezek az adatok később nem módosíthatóak a tulajdonos számára. Ha változtatni kell, akkor azt csak az teheti meg, aki kiadta neki a gyűrűt.

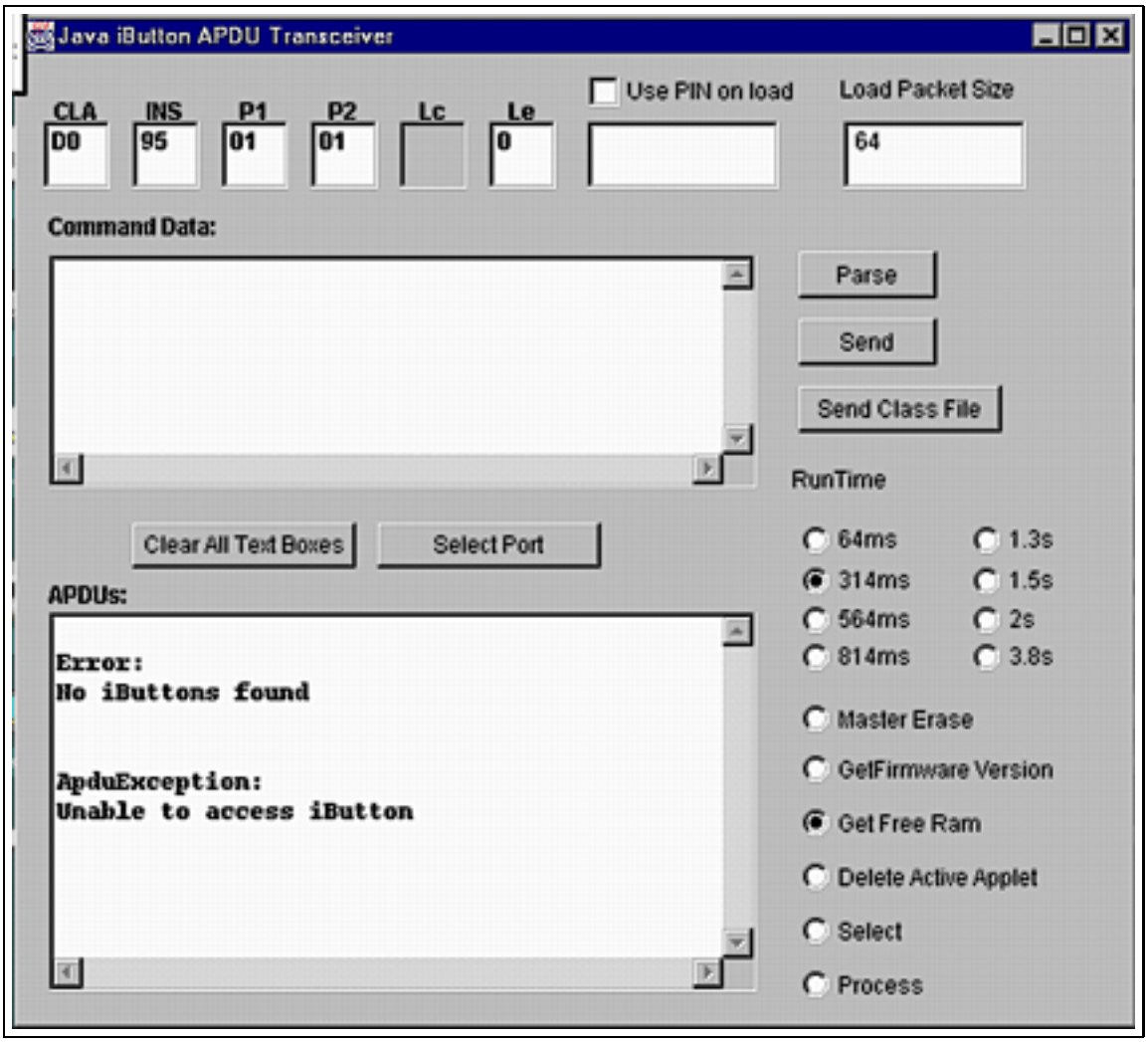
3.2.4. Létező rendszer üzemeltetése, napi használata

Mint már említettem a rendszer áll központi gépből, terminálokból és persze programokból, melyek ezeken futnak. Üzemeltetés alatt ezek karbantartását, figyelését is értem. A rendszert ugyan magára lehet hagyni, de optimális eset ha állandóan figyeli azt valaki, hiszen bármikor bármi elromolhat, és az ilyen hibákat jobb minél előbb kijavítani.

A rendszert fel kell készíteni speciális esetekre. Ha például tűzvész van, akkor minden kifelé vezető ajtót ki kell nyitni, és mellőzni kell minden fajta ellenőrzést. Az ellenkezője is megvalósítható kell lennie, amikor minden ajtót zárni kell, és csak egy-két nagyon kiváltságos üzemeltető számára nyílhatnak ki a kapuk.

A karbantartó szemszögéből a feladat a következő:

- Követi a forgalmat terminálokra lebontva.



apduGUI program. *Ez egy Dallas Semiconductor Corp. által szolgáltatott program. Ennek segítségével közvetlenül tudunk a gyűrűről információt kapni, gyűrűre class file-t letölteni.*

- Ha kell vissza tudja nézni, hogy mely terminálokon kik mentek át.
- Le tud zárni részeket. pl.: Az épület egy területére veszélyes anyag került, akkor blokkolja azokat a terminálokat, melyek megengednék az odajutást.
- Teljesen szabaddá tehet részeket. pl.: Több részből álló épületegyüttes egyikében ha tűz üt ki, akkor ott mindent ajtót ki kell nyitni.
- Az úgynevezett log file-okról készíthet másolatot. Ezek a log file-ok később felhasználhatók arra, hogy visszanézzék egy-egy személy mikor merre járt. Ez lopás esetén hasznos.

Egy felhasználó szemszögéből a rendszer használata a következő:

- Odamegy egy terminálhoz, hozzáérinti a gyűrűjét (iButton) a leolvasóhoz (Blue Dot) majd megkapja a választ, hogy átmehet-e vagy sem. Abban az esetben, ha van PIN kód, vagy ujjlenyomat ellenőrzés akkor még be kell írnia a PIN kódját, vagy a megfelelő ujját kell a leolvasóhoz érintenie.
- Ha valaki nem tudja a pontos utat, ahova mennie kell, akkor segítséget kérhet: a rendszernek van ilyen szolgáltatása, hogy megmondja adott csúcsból a kért csúcsba (terminálótól – terminálig) milyen út (általában a legrövidebb) vezet.

4. Összefoglalás

Adott volt a feladat: Beléptető rendszer készítése előre meg nem határozott objektumra Java Gyűrű segítségével.

A feladat megoldása során felhasználtam sok mindent eddigi tanulmányaimból. Java programozási nyelvben írtam az alkalmazásokat, és felhasználtam a gráfelméleti ismereteimet. Felmerültek azonban teljesen új problémák is. A Smart Cardok, azon belül a Java Kártya (konkrétan az ennek specifikáció szinten megfelelő Java Gyűrű) újdonság volt számomra. Ez adódik abból, hogy a számítástechnika ezen ága viszonylag új. Megismertem ezek működési elvét, előnyeit. Talán egyik legnagyobb előnyük, hogy akár ugyanaz a gyűrű több mindenre is használható, egyszerre több alkalmazás is rátölthető, kibocsátásuk nem végleges. Elképzelhető, hogy egyazon gyűrűn tárolom az egészségügyi adataimat, fizetek fele benzinkútnál, használom egy beléptető rendszernél azonosításra. Egy gyűrűn lévő alkalmazás módosítható, így ha abban hibát fedeznek fel, akkor nem kell az egészet kidobni, csak az új, kijavított alkalmazást rátölteni a régi helyett. Ha ilyen kártyákat használnának a GSM rendszerekben SIM kártyák helyett, akkor most nem lenne baj, valószínű nem kellene az összes kártyát visszavonni.

Ugyanakkor észre kell venni a fejlődés lehetőségét, szükségességét. Elsősorban a memóriakapacitás növelése lenne fontos. Ez az általam írt beléptető rendszerben is használható lenne, mert tárolhatnám a teljes gráfot a gyűrűn, és akkor a gyűrű

maga dönthetné el, hogy az adott kapun átmehet-e a felhasználó, vagy sem. Sokkal kisebb lenne az adatforgalom a központi számítógép és a terminálok között.

Örvendetes lenne, ha az ezzel a témával (Smart Card, Java Card) foglalkozó szakdolgozatok, írások hatására az egyetemek, felsőoktatási intézmények támogatnák ezt a területet, az ilyen irányú kutatásokat akár oktatás szintjén is.

Forráslista és Irodalomjegyzék

A dolgozatomban használt információk egy része a következő nyomtatott illetve elektronikus forrásokból származnak:

Elektronikus:

– http://www.dalsemi.com Dallas Semiconductor Corp. honlapja
– http://www.javasoft.com Java technologia honlap

Nyomtatott:

– Csepregi, Csizmazia, Czabala, Dezső, Fodor, Fhroner, Gulyás, Kozsik, Nyékyné, Porkoláb, Salamon, Szabó, Takács: Java 1.1 Útikalauz programozóknak, Kalibán BT, ISBN 963 04 7417 4
– Cormen, Leiserson, Rivest: Algoritmusok, Műszaki Könyvkiadó, ISBN 963 16 1389 5