SZAKDOLGOZAT

Buda Sándor Péter

Debrecen
2008

Debreceni Egyetem

Informatikai Kar

JÁTÉKFEJLESZTÉS MOBILKÉSZÜLÉKRE

Témavezető:
Bátfai Norbert
egyetemi tanársegéd

Készítette:
Buda Sándor Péter
programtervező informatikus

Debrecen
2008

Tartalomjegyzék

Ábrajegyzék

1. Bevezetés

Mivel a mai mobiltelefonok nagy része nemcsak beszélgetések lebonyolítására alkalmas, hanem számos egyéb funkcióval is rendelkezik. napjainkban talán nem is lehet még a legolcsóbb mobilkészülék között sem olyant találni, amely ne rendelkezne játékokkal, illetve csaknem mindegyikre letölthetőek új játékok, melyek nagy közkedveltségnek örvendenek a készüléktulajdonosok körében.

A kezdetben unaloműzésre kitalált egyszerű játékprogramoknak mára számtalan bonyolult típusa és változata került forgalomba, komoly bevételt jelenthet készítőik számára. Beszerzésük legegyszerűbb módja a mobilszolgáltatók WAP-portáljáról való letöltésük, melynek elszámolása a havi számlán történik, illetve a feltölthető kártya esetében az adatátvitel költsége automatikusan levonódik az egyenlegből. A játékok fajtái, árai és az, hogy milyen készülékekre tölthetőek le, mobil-szolgáltatónként eltérhetnek.

A mobiljátékok népszerűségét hazánkban nagy mértékben fokozta, hogy az elmúlt évek során Magyarországon is elterjedt a színes kijelzős készülékek használata, mely látványosabb szórakozást tett lehetővé a felhasználók számára. A legközelebbi lépés a mobiljátékok piacán a következő generációs készülékek megjelenése, melynek bevezetése már Magyarországon is folyamatban van. Ezekkel a készülékekkel ekkor már térben egymástól távollévő játékosok is részt vehetnek közös játékban.

Manapság egyre gyakoribbak azok a mobiltelefonok, amelyek a bennük lévő gyors processzorok és a beépített, vagy kártyaként behelyezett nagy kapacitású memóriáknak a hatására képesek nagy mennyiségű, bonyolult számítást igénylő programok futtatására. A megnövekedett forráskódok és felhasználói igények arra késztették a fejlesztőket, hogy olyan technológiákat alkalmazzanak, amelyekkel egyszerűbben és gyorsabban lehet alkalmazásokat fejleszteni. Megjelentek a Java-t támogató telefonok, ezeken a Java egy speciális verziója a J2ME, amely jelenleg is a legelterjedtebbek közé sorolható, de vannak olyanok is amelyek más operációs rendszert futtatnak mint pl. a Symbian OS. Ezek a technológiák az szoftverfejlesztési ágazaton belül, a játékszoftver készítésnek is kedveznek. A PC-re, a különböző típusú konzolgépekre ugyanúgy mint a mobiltelefonra nagyon rövid időn belül jelennek meg új játékok, melyet természetesen a felhasználók azonnal elkezdenek

fogyasztani, de az is tény lehet, hogy valaki azért cseréli le PC-jét vagy mobilját, mert a legújabb játék már nem fut el a „vason”. Ezek a széles kínálati paletta miatt nemcsak a fiatalabb korosztályt is vonzzák, hanem akár a felnőtteket is.

A mobilos játékok fejlődése szinte megállíthatatlan. Napjainkra már annyira felgyorsult, hogy szinte gomba módra jelennek meg rövid időn belül az újabbnál újabb játékok. Persze az újítás a játékiparban nem mindig éri meg a befektetett energiát. Viszont azt el lehet mondani, hogy az eddigi újító szándékú fejlesztők akár sikerét, akár kudarcát tekintve, alapjában véve hasznára válik a mobilvilágnak, ha a fejlesztők, mernek újat mutatni, hiszen akár jó is származhat belőle, ha pedig nem, akkor azzal is színesítették a mobiljátékok palettáját. Napjainkban óriási versenyről van szó, aminek igazi nyertesei a felhasználók, akik nélkül a fejlesztés nem tartana ott, ahol ma. Mindenképpen nagy a szerepük mind a játékok fejlődésében, mind pedig a mobilfejlődést általában véve, hiszen az alkalmazások nekik készülnek, az ő szórakoztatásukra, az igényeiknek megfelelően.

A dolgozatom célja nagyvonalakban bemutatni „Mobilvilág” robbanásszerű fejlődését, a Java technológia milyen mérföldköveken haladt keresztül, mennyire sokrétű ez a platform, aminek a gyors fejlődési sikerei lehetővé tették mind a technológia, mind az azt kihasználó üzleti lehetőségek világméretű elterjedését és hogyan sikerült megvalósítani egy Java 2 Micro Edition gyakorlati alkalmazást, amiben bemutathatom a J2ME-ben rejlő lehetőségeket. Ez nem más mint egy mobiltelefonon is játszható, hálózati kapcsolatot is megvalósítható felhasználó játék megalkotását célozta meg, melynek eredményeképp született meg a Kalózok nevű program. A játék ugyan csak ún. demó verzióban létezik, és hálózati kapcsolatot egyenlőre nem használ, de a Java programozási nyelv tulajdonságainak és a NetBeans felhasználóbarát kezelésének köszönhetően kisebb módosításokkal, könnyen teljes értékű alkalmazássá tehető.

2. A mobilról általában

A hagyományos telefonrendszer, még ha egy szép napon több gigabites fényvezető szálakkal is fog rendelkezni a végpontok közötti teljes szakaszon, akkor sem fogja tudni kiszolgálni a felhasználók egy bizonyos, egyre növekvő csoportját. Ez a csoport pedig az utazó, folyton úton levő felhasználóké. Az emberek manapság elvárják, hogy

telefonálhassanak repülőről, autóból, uszodából és az esti kocogás közben is. Többek között azt is, hogy ezekről a helyekről és máshonnan is küldhessenek elektronikus leveleket, továbbá a Világhálón is szörfölhessenek. Ezáltal a vezeték nélküli telefonálás iránt hatalmas méretű az érdeklődés.

A mobiltelefon-előfizetők száma idén év végére világszerte el fogja érni a négymilliárdot. Az előfizetések száma az utóbbi 8 évben évente 25 százalékkal emelkedett. Míg a mobiltelefonok elterjedtségi mutatója 2000-ben csak 12% volt, 2008 végére már meghaladja a 60%-ot. A mobiltelefonos előfizetések száma régiónként és országonként lényegesen eltéréseket mutat. A legnagyobb arányú növekedést az olyan gyorsan fejlődő országokban rögzítették, mint Brazília, Oroszország, India és Kína. Csak ebben a négy országban az év végére 1,3 milliárd mobil-előfizető lesz. Statisztikai jelentés a KSH illetve az NHH szerint a 2008 augusztus végén, Magyarországon lévő hívásfogadásra képes aktív SIM kártyák száma 11.688.471.

Európában, - ellentétben Amerikával - nagy hatással van a mobiltelefonok terjedésére, az előre fizetett un. felöltőkártyás mobiltelefonok széles körű használata (bizonyos területeken akár meg is haladja az előfizetésesek számát). Ezeket sok üzletben ugyanazokkal a formaságokkal lehet megvásárolni, mint egy bármilyen más fogyasztási cikket. Előre fel vannak töltve egy bizonyos lebeszélhetőségű összeggel, jár hozzá egy titkos PIN kód és újra fel lehet tölteni, amikor a felhasználó úgy gondolja. Ebből kifolyólag Európában gyakorlatilag minden tizenéves és néhány ennél fiatalabb gyerek is rendelkezik (általában felöltőkártyás) mobiltelefonnal talán azért, hogy szüleik könnyebben ellenőrizhessék őket vagy csak azért mert egyszerűen ez a trend és ha „nekem nincs de az osztálytársnak van”, akkor rögtön konfliktus forrása lesz. Mindezt ráadásul anélkül, hogy a gyerek óriási számlát tudna csinálni. Ha a mobiltelefont csak ritkán használják, akkor lényegében ingyen van, mivel nincs havi előfizetési díj, és a bejövő forgalmat sem számlázza ki a szolgáltató. Az igény óriási, tehát egyre többen akarnak ilyen jellegű „köldökzsinórt” a jelenlegi információs társadalmunkban. Az más kérdés, hogy mennyire lehet felügyelni, kontrollálni illetve tudatosan igénybe venni, kihasználni ezeknek a készülékeknek és szolgáltatásoknak a lehetőségeit.

Az UNICEF munkájának eredményeképpen az ENSZ közgyűlés 1989-es gyermekek jogairól szóló elfogadott határozata így szól: „...a gyermeknek joga van az érettségének, értelmi és

lelki fejlettségének megfelelő információkhoz jutni, ugyanakkor meg kell védeni őt a nem megfelelő tartalmak elérésétől...”

A mobiltelefonról alkotott kép még a profi felhasználók számára is túl gyorsan változik. Egy új telefon megjelenésével a korábbi készülék hirtelen elavulttá válik. Az egyre kifinomultabb szolgáltatások kitalálását nyilván mindig a modernebb mobiltelefonok ösztönzik, vagyis a kis eszköz nagy dilemmák tárgyává válik az üzleti és magánfelhasználásban egyaránt. Új fejlesztő és szolgáltató generációk készülnek arra, hogy kizárólag erre a készülékre, azaz a mobiltelefonra alapozzák jövőjüket, karrierjüket.

2.1. Egy kis „mobil” történelem

2.1.1. Az első generáció - „Fapados” korszak

Az első mobiltelefon-rendszert az AT&T dolgozta ki Amerikában. A mobiltelefonok a hálózatok folyamatos fejlesztése révén (az 1980-as évek második felében) kezdtek el rohamosan terjedni. A cellás rendszerű hálózatok átjátszó állomásai eleinte nagyon közel helyezkedtek el egymáshoz és problémát jelentett, ha a mobil egyik cellából a másikba lépett.

Ebben az időben a gyártók analóg rendszereket használtak, a készülékek méretei is jóval nagyobbak voltak és szinte csakis autóba építették őket, így születtek meg az autóstelefonok. Az első ilyen modelleket a mikroelektronikában élen járó, svájci székhelyű Natel (Nationales Autotelefon) hozta forgalomba. Ezután a technika fejlődésével készítettek aktatáska méretű telefonokat, majd a Motorola jóvoltából elkészült az első kereskedelmi forgalomba hozott mobiltelefon, a DynaTAC 8000X 1983-ban.

1981 szeptemberében Szaúd-Arábiában építették ki az első automata roamingos NMT (Nordic Mobile Telefony) hálózatot, melyet a svéd rádió (SRA - Svenska Radio Aktiebolaget) épített ki. Pár hónappal később Skandináviában is beindították az NMT szabványon alapuló hálózatot, amely először 150 MHz-es frekvencián, majd 1986-ban már 900 Mhz-en működött. A Nokia és az Ericsson ebben az időszakban mindent megtettek, hogy piacvezető gyártókká váljanak.

2.1.2. A második generáció - GSM hálózatok

Az második generációs hálózatok fejlesztése a '90-es évek elején kezdődött meg. Az első digitális mobiltelefont 1990-ben mutatták be az Egyesült Államokban. Európa erre egy évvel később válaszolt a GSM-szabvány bevezetésével. A második generációs hálózatok általános jellemzője a korábbinál gyorsabb és erősebb hálózati kapcsolódás illetve a digitális technológiák erőteljes alkalmazása.

Európában általánossá vált a 900 MHz-es frekvencia használata az 1G és 2G készülékek esetében, viszont az első generációt igyekezték minél hamarabb leépíteni, hogy helyet biztosítsanak a GSM rendszerek fejlesztéséhez. Amerikában és Kanadában viszont továbbra sem mondtak le az analóg technológiáról, amelyek más frekvenciákat preferáltak.

Ebben az időszakban a gyártóknak módjuk volt a telefonok méretének csökkentésére (innen származik a találó "féltégla" elnevezés), fejleszteni a akkumulátorokat, energiatakarékosabb alkatrészeket építettek be és igyekeztek minél több hasznos funkcióval felvértezni modelljeiket.

A 2,5G (2,75G) hálózatok a második generációra épülve nagyobb sebességű adatátvitelt tettek lehetővé, ide tartozik például a GPRS és az EDGE.

1997-ben az Iridium műholdak fellövése. Műholdas telefonrendszer nem kifizetődő vállalkozás, egyenes út 1999-ben a bukáshoz. 2001-ben viszont újraindult a szolgáltatás.

2.1.3. A harmadik generáció - 3G Nincs megállás

A korábbi módszerektől eltérően a harmadik generációtól kezdve már nem a meglévő alapokra épülnek a hálózatok. Más a működési frekvencia és kifejezetten a nagy sebességű adatátvitelre és a multimédiás szolgáltatások szinte végeláthatatlan sorára koncentrálnak a fejlesztők. Az első kereskedelmi 3G-s (CDMA) hálózatot Japánban mutatták be 2001-ben. Európában ugyanez csak 2003-ban következett be. Becslések szerint jelenleg 60-féle 3G hálózat működik világszerte.

2004-ben a Kiotói és a Pekingi Egyetem, valamint a Fujitsu együttműködésével elindulnak az első 4G-s hálózati fejlesztések. A világ GSM-felhasználóinak száma meghaladja az egymilliárdot.

2.1.4. A negyedik generáció - 4G Jelenből a jövőbe

A 3G-nél persze nem áll meg a technológiai fejlődés, hisz most bontogatja szárnyait a HSDPA, HSUPA és nemcsak külföldön hanem hamarosan hazánkban is a 4G-s hálózatokat tesztelik. A Samsung a 4G-s hálózat tesztelésekor 100 Mbit/s adatátviteli sebességet ér el egy mozgó járművön és 1 Gbit/s sebességet sétálás közben.

Az új Huawei HSDPA rendszer lehetővé teszi a 14,4 Mbit/s névleges letöltési sebességet is, melyet a Huawei és a Vodafone már a világon több helyen is tesztel. A sikeres együttműködés hátralévő feladata a 14,4 Mbit/s mobilinternetes letöltési sebesség magyarországi tesztelése, amely 2008. október végétől kezdődik. Világszerte több mint 2 milliárd a mobil-előfizetők száma.

Alternatív megoldásként igyekeznek WiBRO, WiMAX, WLAN, Wi-Fi és egyéb módszereket is alkalmazni a jövő mobiltelefonjainál.

Adatátviteli sebességek: a fejlődés tükörképe

CSD(GSM – 2G).....14,4 kbit/s
HSCSD(2,25G).....43,2 kbit/s
GPRS(2,5G).....55,6 kbit/s
EDGE(2,75G).....236,8 kbit/s
UMTS(3G).....384 kbit/s
HSDPA(3,5G).....3,6-14,4 Mbit/s
HSUPA(4G).....~0,1-1 Gbit/s

3. Java és a mobil

3.1. Gondolatok a Java-ról

Mi is ez a szó, amely annyiszor hagyja el korunk emberének száját, de tudja-e, hogy valójában mit is takar? Kérdés, hogy honnan akarjuk ezt megközelíteni, mert több oldalról is lehetséges. Ez egy sziget, avagy egy kávé, esetleg programnyelv, platform, technológia? Attól eltekintve, hogy vannak kapcsolódási pontok ezen különböző objektumok között, természetesen én az utóbbiakra gondoltam. A Java egy objektumorientált programozási nyelv, amelyet a Sun Microsystems fejleszt a 90-es évek elejétől kezdve és ez a folyamat napjainkban is tart. Eredetileg a Java programozási nyelvet James Gosling és egy mérnökökből álló csapat hozta létre a Sun Microsystems-en belül. A fejlesztés 1991-ben kezdődött a Green Project részeként. Hivatalosan 1995. május 23-án jelentették be, az első verziót novemberben adták ki. Eredetileg a C++ utódjának szánták. Az eredeti neve Oak volt (ami angolul tölgyfát jelent, állítólag a James Gosling irodája előtt álló fa ihlette), azonban ez már egy bejegyzett név volt, ezért végül Java néven vált ismertté. A Java szó a Sun Microsystems védjegye.

A fejlesztés során előforduló fontosabb mérföldkövek:

- JDK 1.0, 1996. január 23.
- JDK 1.1, 1997. február 19.
- JDK 1.2, kódneve *Playground* (más néven Java 2), 1998. december 8.
- JDK 1.3, *Kestrel*, 2000. május 8.
- JDK 1.4, *Merlin*, 2002. február 6.
- JDK 5.0, *Tiger* (korábbi nevén 1.5), 2004. szeptember 30.
- JDK 6, *Mustang*, 2006. december 11.
- JDK 7, *Dolphin,* folyamatban (JDK 7 – b39 releases 2008. november 6.)

3.2. A Java mint technológia

A Java technológiát nevezhetjük egyaránt programozási nyelvnek és platformnak.

3.2.1. Programozási nyelv

A Java egy magas szintű nyelv, amelynek a következő fő jellemzői vannak:

Egyszerűség

Nem kifejezetten kezdésnek lehetne ajánlani, de a nyelv, valójában a C++ leegyszerűsített változatának tekinthető, tehát nem okoz különösebb „fejfájást” annak, aki programozói ismeretekkel rendelkezik. Ellentétben a C++-vel, nincs többszörös öröklődés, ami az OO nyelvek egyik hatékony tulajdonsága. Elkülöníti a típus definíciójának és megvalósításának öröklődését. Az interfészeken keresztül ugyan engedélyezi a típusdefiníciók többszörös öröklődését, de a megvalósítások csak egyszeresen örököltethetők. Ezzel kihasználható a többszörös öröklődés számos előnye, sok veszély pedig elkerülhető. Nincs benne ugró utasítás, van viszont valósidejű kivételkezelés és a már nem használt memóriaterületek automatikus felszabadítását végző „szemétgyűjtés”.

Objektum orientáltság

Az objektum orientáltság egy programozási szemléletmód és egy nyelvi jellemző. Fő gondolata az, hogy a tervezés középpontjában azok az objektumok álljanak, amikkel a program dolgozik, és ne a műveletek, amiket végrehajt.

Platform függetlenség

A platform függetlenség azt jelenti, hogy a Java nyelven írt programoknak hasonlóan kell futniuk különböző architektúrájú hardvereken és operációs rendszereken. Ha egy programot egyszer megírunk valahol, akkor az mindenhol futtatható kell legyen. Ezt úgy lehet elérni, hogy a Java nyelvű kódot egy köztes nyelvre, az ún. bájt-kódra fordítom le, ami virtuális gépi utasításokból áll. Ezt a kódot ezután egy Java virtuális gépen futtatom, ami egy olyan program, ami az adott számítógép gépi kódjára van lefordítva, a feladata pedig az, hogy a Java bájt-kódú utasításokat a helyi gépen használható utasításokra fordítsa le.

Többszálúság

A programok nagy része vezérlési szálakra bontható a processzor,(-ok) jobb kihasználhatóságának érdekében és ez teszi lehetővé a párhuzamos végrehajtást. A többszálú párhuzamos programozáshoz szükséges kölcsönös kizárást a Java nyelvi szinten oldja meg. A Thread osztály az ami a szálak létrehozásához szükséges szinkronizációt tartalmazza. Az egymással kommunikáló szálakra bontott feladat könnyen áttekinthető és a számítógép jobb kihasználtságát eredményezi.

Dinamikusság és hozzáférhetőség

Továbbfejlesztése könnyű, osztálykönyvtárak szabadon bővíthetőek. A hozzáférhetőség az a fontos tulajdonság, amely az elterjedésében fontos szerepet játszott. A fejlesztői környezettől kezdve rengeteg példaprogram, forráskód, specifikáció és dokumentáció ingyenesen elérhető. Mivel elég sok támadás érte a Sun-t a zárt forráskódú Java eszközök miatt, ezért 2007-ben a Sun Microsystems bejelentette, hogy kiadják a Java platformot, a Java programozási nyelv és környezet referenciamegvalósítását a GNU General Public License (Általános Nyilvános Licenc) szerződés feltételei mellett, ami meg is történt.

Biztonságosság

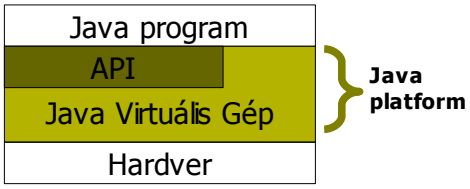
A Java platform az első olyan rendszerek egyike, ami támogatta a távoli helyen lévő kódok futtatását. A nyelv tervezésekor nagy figyelmet szenteltek a hálózatkezelésnek. Egy applet futhat a felhasználó böngészőjében egy olyan kódot végrehajtva, amit egy távoli HTTP szerverről töltött le. Ez a távoli kód, egy szigorúan felügyelt és korlátozott környezetben fut, így védi a felhasználót a hibásan működő vagy rosszindulatú programoktól. A szerző digitálisan aláírhatja az appletjét, így jelezve, hogy az biztonságos, ezzel a program olyan jogokat kaphat, amivel kiléphet ebből a környezetből és például elérheti a helyi fájlokat, használhatja a hálózatot.

3.2.2. Java platform

A platform, hardver vagy szoftverkörnyezet, ahol a programok futnak. A legtöbb platform a hardvert és az operációs rendszert jelenti. A Java platform annyiban különbözik a legtöbb mástól, hogy teljesen szoftverplatform, és más hardver alapú platformokra épül.

A Java platform két fő komponensre különíthető el:

- ✔ Java VM (Virtual Machine)
- ✔ Java API (Application Programming Interface)



1. ábra: Elszigetelés

Az ábra bemutatja a Java platform működését, hogyan szigeteli el az API és a JVM, a Java programot, az „alatta” lévő hardvertől.

3.2.2.1. Java VM – Java Virtuális Gép

Amikor egy számítógépen elindítunk egy programot, akkor a program utasításait - amelyek megmondják hogy a processzornak lépésről lépésre mit kell csinálnia - eleve olyan formában tárolják, amelyet az adott processzor közvetlen megért. Ezt nevezik natív futtatható állománynak. A natív állományok igen nagy hátránya, hogy ez a fajta kód nem hordozható. Ha például egy RISC processzorra lefordított alkalmazást megpróbálunk elindítani egy CISC utasításkészletű processzorral szerelt gépen, akkor ez természetesen nem fog működni. A virtuális gép egy újabb réteget definiál a gép natív platformja illetve a Java program közé. Ez lényegében egy futtató környezet, amelyet előre kell telepítenünk a gépen ahhoz hogy a Java programokat futtatni tudjuk. A Java virtuális gép az úgynevezett bájt-kódot képes értelmezni, vagyis az egyes Java programokat ilyen binárisba fordítja a fordító. A bájt-kód egy átmenet a natív kód és a forrásfájl között.

Joggal kérdezhetném, hogy mi értelme a virtuális gépnek? A hordozhatóság. Egy Java-ban megírt programot lefuttathatunk bármely platformon amely rendelkezik Java virtuális géppel. Pl. egy Windows alatt lefordított java alkalmazást simán lefuttathatunk Linuxon is, ha mindkét helyen telepítve van Java virtuális gép. Gondoljunk csak bele, virtuális gép nélkül lehetetlen lenne kivitelezni olyan weblapokba épülő kisalkalmazásokat, amelyek bármely architektúra bármely böngészőjén futnának, módosítás nélkül. A Java Applet-ek viszont képesek erre.

3.2.2.2. Java API- Alkalmazásprogramozási Felület

Egy program vagy rendszerprogram azon eljárásainak, szolgáltatásainak és azok használatának dokumentációja, amelyet más programok felhasználhatnak. Egy nyilvános API segítségével lehetséges egy programrendszer szolgáltatásait használni anélkül, hogy annak belső működését ismerni kellene.

Az API általában nem kötődik programozási nyelvhez: bármilyen programnyelvből lehetséges azok meghívása, amennyiben a megfelelő paramétereket a hívás biztosítja, és képes lekezelni az esetleges eredményt. A Java API igen sok (több ezer) használatra kész szoftverkomponenst tartalmaz: csomagokba szervezett osztályokat és interfészeket.

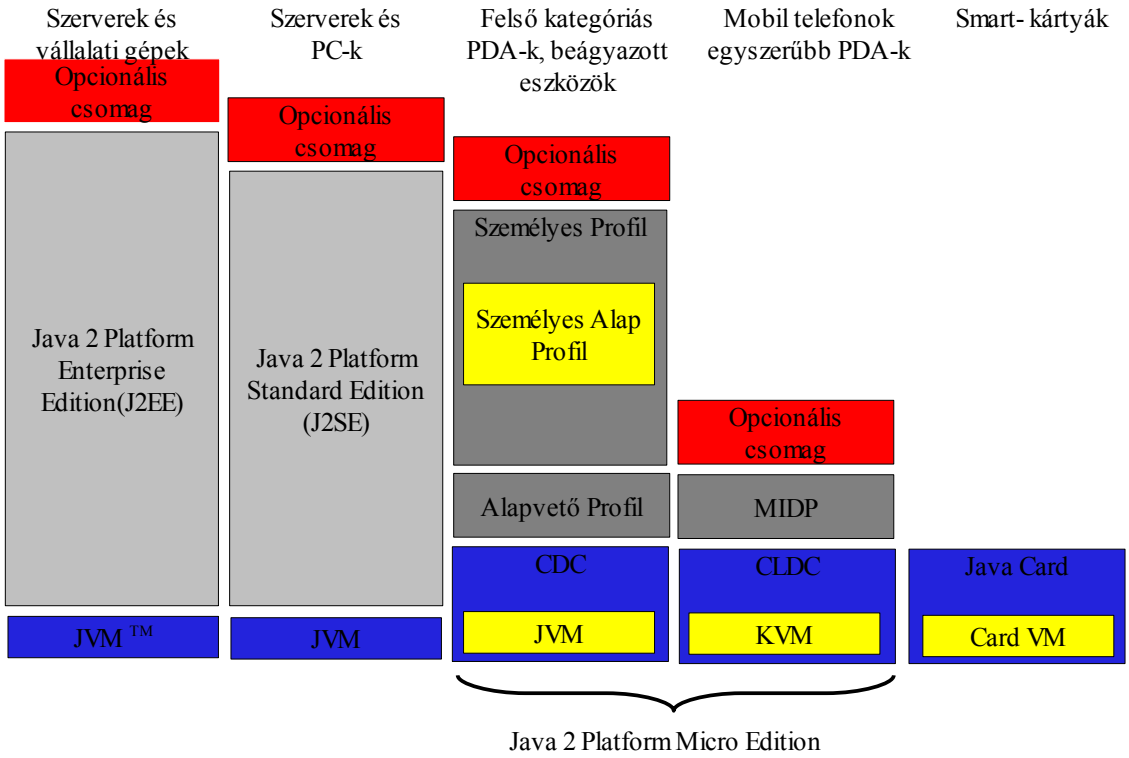
Még mielőtt a Java platformot tovább boncolgatnám először is szeretném tisztázni ezeket a kifejezéseket, hogy servlet, applet, midlet.

Servlet - Egy speciális program. Szerver oldalon fut, de nem önállóan, hanem egy szerver futtatókörnyezet részeként. Pl. egy portált ki lehet szolgálni néhány servlet együttesével, vagy akár egyetlen servlet-tel is.

Applet - Olyan asztali alkalmazás, program, amely közvetlenül a Java platformon futtatható. Ez bizonyos megszorításokkal futtatható Javát ismerő böngészőben és ezt látványos grafikai effektusok készítésére használják.

Midlet - Így nevezik a kézi számítógépeken, mobil telefonokon futó alkalmazásokat.

Ezek után könnyebb lesz megfogalmazni a Java verzióit amelyek valamilyen általánosságban lefednek egy-egy felhasználási területet. Maga a nyelv nyilván az igényeknek megfelelően szűkítve esetleg bővítve lett, hiszen nem mindegy hogy egy Java programot egy többprocesszoros szervergépre, vagy egy kis mikró rendszerre fejlesztettek. Itt nem csak a teljesítményre, hanem az egyes lehetőségek szűkülésére és bővülésére is gondolni kell. A Sun a teljes káosz elkerülése végett, bevezette az ún. Java platformokat.



2. ábra: Java verziók

- **Java EE** (Enterprise Edition): nevéből adódóan üzleti alkalmazásra szánt változat, gyakran nagyon erős, többprocesszoros szervergépekhez, gigabájtos méretű memóriával. Leginkább webalkalmazások fejlesztésére használják. Fontos eleme a servlet, amely egy olyan kis java alkalmazás, amely egy kliens felőli kérést hivatott kiszolgálni. Fontos még megemlíteni a JSP-t (Java Servlet Pages), amely hasonlóan az ASP vagy PHP nyelvekhez dinamikus oldalak előállításában vesz részt.

- **Java SE** (Standard Edition): kifejezetten munkaállomásokra szánt változat. (pl. PC-re) Tulajdonképpen a Java itt indult az Applet-ekkel, majd az „önálló” asztali alkalmazások is teret hódítottak. Ennek a platformnak kellően nagy memóriája és processzora van, illetve fontos a felhasználó számára kényelmes felhasználói felület is.

- **Java ME** (Micro Edition): elsősorban telepes üzemű, kis kijelzőjű, korlátozott beviteli lehetőségekkel és processzor teljesítménnyel rendelkező eszközökre fejlesztették ki pl.: telefonok, PDA-k, személyhívók. A J2ME az ugyanilyen célból készült Personal Java utódja (teljes mértékben le is váltotta azt). A J2ME virtuális gépe a KVM (Kilobyte Virtual Machine). Nevéből adódóan kilobyte-os méretű, kifejezetten a mobil környezethez íródott, és igényeknek megfelelően modulárisan bővíthető. A J2ME profilban az

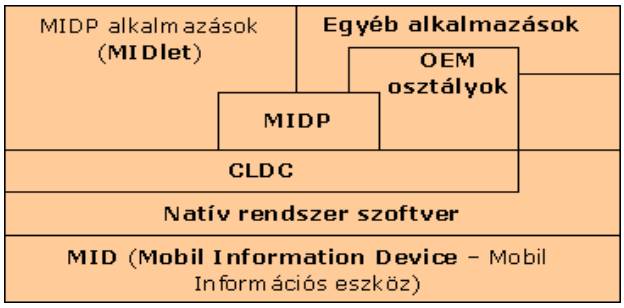
alkalmazások kezeléséért és telepítéséért a JAM (Java Application Manager) felelős. Az egyes alkalmazásokat jar fájlok képében telepíthetjük.

- **Java Card** (Smart Card): Kis teljesítményű processzorral és kevés memóriával rendelkező eszközökre való fejlesztés (intelligens kártyák). Digitális azonosítási megoldások, ahol elvárás, hogy a különböző gyártók kártyáival képesek legyenek együttműködni az alkalmazások.

3.3. Java ME - Java Micro Edition

Ez a Java Platform, flexibilis környezetet nyújt olyan alkalmazások számára, melyek mobiltelefonon és egyéb beágyazott készülékeken - mobiltelefonok, személyi digitális asszisztensek (PDA-k) futnak. A Java ME egyaránt tartalmaz rugalmas felhasználói felületet, biztonsági mechanizmusokat, beépített hálózati protokollokat és támogatást a hálózati és az offline alkalmazások számára. A legkisebb komponenst is további részekre kell tagolni, hiszen a J2ME platformon belül is bevezettek különböző profilokat, hogy azok még inkább illeszkedjenek egy-egy hardver típusra. Minden ilyen profil alapja az úgynevezett CLDC - Connected Limited Device Configuration (Csatlakoztatott Limitált Eszköz Konfiguráció). A CLDC konfiguráció tartalmazza a J2ME alap osztálykönyvtárát (java.lang csomag szűkített változata), így az alapvető Java nyelvhez tartozó elemeket (pl. tömb, karakterlánc, kivételkezeléshez szükséges osztályok), az IO kommunikációhoz szükséges osztályokat (kibővítve a mobil környezet igényeivel) illetve az util osztály egy szűkített készletét. A CLDC közvetlen a hardver natív rendszer szoftvere felett áll jól látszik ez a az architektúrán. A konfigurációkban egy virtuális gép, illetve minimális könyvtárhalmaz van definiálva.

Ezek biztosítják az egy kategóriába tartozó eszközök számára az olyan alapfunkcionalitásokat, mint a hálózati kapcsolat és memóriakezelés. Az architektúrából világosan kiderül, hogy jelenleg két J2ME konfiguráció létezik:



3. ábra: MIDP architektúra

- ✔ CDC (Connected Device Configuration)
- ✔ CLDC (Connected Limited Device Configuration)

3.3.1. CDC - Connected Device Configuration

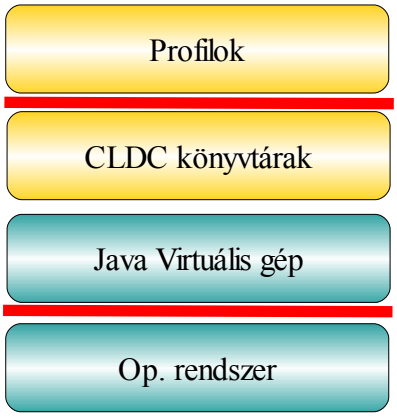
A CDC-t olyan jellegű fejlettebb eszközökre tervezték mint pl. a PDA-k, melyeknek követelményei:

- ✔ 32 bites processzor
- ✔ 2 MB RAM
- ✔ 2 MB ROM
- ✔ operációs rendszer
- ✔ nagyobb hálózati sávszélesség

3.3.2. CLDC - Connected Limited Device Configuration

A CLDC-t olyan jellegű korlátozott erőforrásokkal rendelkező eszközökre tervezték mint pl. a mobiltelefonok, melyeknek a minimális követelményei:

- ✔ 16 bites processzor
- ✔ 192 kB memória, aminek megosztása
  - ◆ 160 kB ROM a KVM és CLDC részére
  - ◆ 32 kB RAM a KVM részére
- ✔ alacsony energiafogyasztású akkumulátor
- ✔ kisebb hálózati sávszélesség



4. ábra: A CLDC architektúra

A CLDC konfigurációra épülő profilok:

3.3.2.1. IMP-Information Module Profile

Olyan eszközökhöz fejlesztették ki ezt a profilt, amelyek egyáltalán nem, vagy csak minimálisan rendelkeznek kijelzővel, viszont valamilyen limitált kétirányú kommunikációra szükségük van. (pl. router-ek, telefonközpontok, hálózati kártyák, árusító gépek)

3.3.2.2. MIDP-Mobile Information Device Profile

Mobiltelefonokhoz fejlesztették ki kifejezetten ezt a profilt. A MIDP profilban az egyes alkalmazásokat MIDlet-eknek nevezik. Legfontosabb bővítése az az LCD API amellyel alapvető GUI felületeket és 2D-s megjelenítést lehet létrehozni (játékhoz ezt fogom felhasználni).

3.3.3. A MIDP profil

J2ME mobiltelefonokra jutó profilját a MIDP képezi. Mint minden mást, a MIDP-t is folyamatosan fejlesztik. A Mobil Information Device Profile-t fejlesztő csoportot a Motorola vezeti a Java Community Process-en belül. A csoportban a Motorola és a SUN Microsystems mellett azonban több tucat mobiltelefon gyártó, mobilszolgáltató, és szoftverfejlesztő is található. Jelenleg a MIDP2 az elterjedt, illetve az ún. Advenced MIDP2, de már folyamatban van a MIDP3 fejlesztése is, így mobil fronton újabb követelményeknek megfelelő eszközök tűnhetnek fel hamarosan.

Tekintsünk be egy kicsit a MIDP változatokba és legfőbb jellemzőikbe:

3.3.3.1. MIDP 1.0 - A múlt

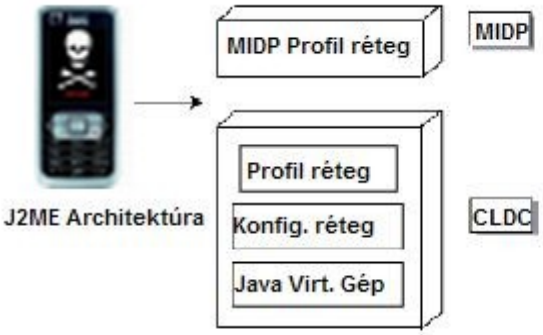
A MIDP változatok közül a MIDP 1.0 az első, amelynek eredeti változatát 2000. szeptember 1.-én adták ki, majd nagyjából három hónappal a kiadás után elkészült a végleges 1.0a (JSR 37) változat is. Ez a szabvány az IT terület fejlődési tendenciájához viszonyítva elégé régi keletű, ám a későbbi profilok visszafelé kompatibilisek.

A MIDP specifikációkat a MIDPEG (Mobile Information Device Profile Expert Group) csoport készítette, amelynek az 1.0-ás változatnál 22 tagja volt (csak néhány a legfontosabbak közül: Nokia, Ericsson, Motorola, Siemens, Samsung). A MIDP 1.0a a mai telefonok képességeihez viszonyítva meglehetősen nevetséges követelményeket támasztott, de nem lehet megfeledkezni róla, hogy ekkor még csak nagyrészt monokróm kijelzős készülékek uralták a piacot, és ezek között is kevés volt Java képességű. A kezdeti időszakban az elterjedésüket akadályozta, hogy csak a nagyon drága, luxus készülékek voltak Java-képesek, és megfelelő alkalmazások hiányában nem is volt a felhasználók részéről igény a látszólag haszontalan szolgáltatás igénybevételére. A specifikáció minimális jellemzői:

- ✔ 96x54 pixel képernyő méret, 1 bites színmélység, nagyjából 1:1 arányú kijelzővel
- ✔ egy-, kétkezes billentyűzet, esetleg érintőképernyő bemeneti eszközként
- ✔ Memória: 128kB ROM a MIDP komponenseknek, 8kB ROM a MIDP alkalmazások számára szükséges hosszú távon tárolandó adatoknak (pl. beállítások), 32kB RAM a KVM halomterületnek (Heap)
- ✔ kétirányú, vezetéknélküli, korlátozott sávszélességű kapcsolat

A MIDP 1.0-ás változatban kerültek definiálásra a MIDlet alapját képező osztályok és interfészek (javax.microedition.midlet), az LCD API (javax.microedition.lcdui csomag), a hosszú távú adatok tárolását elősegítő osztályok (javax.microedition.rms csomag), illetve egy kezdetleges IO csomag, amely ekkor még csak http kapcsolatokat tudott létesíteni (javax.microedition.io csomag).

A MIDP 1.0-nak igen sok hiányosságot felróttak. Többek között, hogy nem támogatta a közvetlen grafikus műveletek elvégzését, így például a megjelenő képpontok színét sem lehet meghatározni, nem lehetett Audio-t programozni, és lekérdezni a billentyűk aktuális állapotát. Később ezeket egyes gyártók kiegészítő csomagokkal próbálták orvosolni. Ilyen pl. a Nokia UI, amellyel már előállíthatóak egyszerű hang-effektusok és a vibra motor is progra-mozható.



5. ábra: A J2ME architektúra

3.3.3.2. MIDP 2.0 - A jelen (ami múlóban van)

Abból is látszik a MIDP 1.0a sikeressége, hogy a középkategóriás telefonok valamivel kevesebb mint felét, a felső kategóriások majdnem egészét látják el Jáva támogatással. A növekvő népszerűség a szabvány bővített változatának elkészítésére késztette a gyártókat, ez a MIDP 2.0 (JSR-118). A MIDP 2.0 végleges változata 2002. november 5.-kén került kiadásra.

A Mobile Information Device Profile 2.0-ás változata számos területen komoly előrelépést jelent a korábbi, 1.0-ás verzióhoz képest. Ami a talán legáltalánosabb mobil Java alkalmazásokat, a játékprogramokat illeti, az új MIDP változat már támogatja a különféle hanghatásokat, és szabványos módszereket kínál a különböző grafikai rétegek, mint az előtérben lévő karakterek, illetve a háttérképek egyszerű és hatékony kezeléséhez. A „Sprite" objektumok támogatása lehetővé teszi, hogy a fejlesztőknek ne kelljen minden egyes képpontot külön kezelniük, hanem a karaktereket és egyéb grafikai elemeket egyszerűen mozgathassák a kijelzőn. Tovább fejlődött a MIDP adattároló rendszere is. A platform általánosan lehetővé teszi, hogy a különféle adatokat a felhasználó készüléke tárolja, szemben a korábbi megoldásnál, hogy az adatokat csak a hálózaton keresztül lehet elérni, egy központi adattárolóról. Az új MIDP változat jelentős előrelépést jelent az internetes szolgáltatások megjelenése felé, és hatékony Web elérést nyújt a mobiltelefonokon keresztül, azaz olyan szolgáltatásokkal rendelkezik, amely az új, fejlett, nagyméretű színes kijelzővel ellátott készülékekkel már élvezetessé teheti a PC-kre szánt weboldalak megjelenítését. Az egyik legjelentősebb újdonság pl. az úgynevezett „push" technológia legújabb változata lehetővé teszi, hogy a készülék egy szerverről bizonyos események bekövetkezésekor információkat, üzeneteket kapjon anélkül, hogy rendszeresen ellenőriznie kellene azok „érkezését", valamint lehetőség nyílik emellett a valósidejű üzenetküldésre is.

A MIDP 2.0 hardver követelményei megegyeznek a MIDP 1.0-éval kiegészítve azzal hogy az eszköznek vagy hardveres, vagy pedig szoftveres úton képesnek kell lennie hangot kibocsájtani (megjelenik a MIDI és a mintavételezett audio támogatás is). Ennek megfelelően megjelent a Media támogatáshoz szükséges osztályokat tartalmazó csomag is (javax.microedition.media). A MIDP 2.0 talán sokkal nagyobb értékű újítása, hogy már tartalmaz olyan osztályokat, amellyel képesek lehetünk biztonságos kapcsolatokat is kezelni

(javax.microedition.pki). Minden MIDP 2.0 eszköz legalább az X.509-es (egyfajta tanúsítvány, hitelesítő szabvány) titkosítást köteles ismerni. Kommunikációnál maradva az 1.0-ás változathoz képest az IO csomag is kibővült olyan osztályokkal, amelyekkel például Socket vagy Stream kapcsolatokat is létesíthetünk a külvilággal.

A MIDP 2.0-t a CLDC 1.1 (JSR-139) konfiguráció tetején tervezték futtatni. Maga a szabvány azonban nem tételez fel a konfigurációról semmit, ami miatt ne lehetne a CLDC 1.0 tetején is futtatni. A CLDC 1.1 tulajdonságai az 1.0-hoz képest.

- ✔ Van lebegőpontos támogatás
- ✔ Vannak gyenge referenciák (egy kis részhalmaza a J2SE gyenge referencia támogatásának)
- ✔ NoClassDefFoundError már van (CLDC 1.0-ban semmilyen Error nem volt)
- ✔ Thread objektumnak vannak nevei
- ✔ Könyvtármódosítások sokasága.

Főként a lebegőpontos támogatás miatt a ROM-igény megnőtt, 160 kBájtról 192 kBájtra.

Bővített biztonsági rendszer a MIDP 2.0-ban

A MIDP 1.0 nagyon egyszerű biztonsági rendszerrel rendelkezik: a MIDlet-ek bármit megtehetnek, de csak bizonyos kereteken belül. Nem férhetnek hozzá semmilyen szolgáltatáshoz, mellyel kár okoznának a telefonban ill. annak használójának. Ez egy egyszerű és biztonságos model, de nagyon korlátozott. A MIDP 2.0-ban egy új, engedélyeken alapuló rendszert vezettek be, ami a J2SE biztonsági rendszerének egyszerűsített változatának tekinthető és teljesen kompatibilis a MIDP 1.0-val.

A MIDP 2.0 midlet sorozatokat aláírják. Ez azt jelenti, hogy a digitális aláírást és a hozzátartozó igazolást (certificate) egyaránt elhelyezik az alkalmazásleíróban (JAD fájl). Érdekesség, hogy a JAR fájloknak van egy aláírási szabványa, azonban a MIDP 2.0 nem ezt használja, a MIDP 2.0 JAR fájlja nincs aláírva. Amikor a JAR letöltődik, a JAR-ra újraszámolják a digitális aláírásban levő hash-t. Ha ez illeszkedik, akkor a megfelelő igazolást az alkalmazásleíróban ellenőrzik, hogy a telefonban tárolt mesterigazolásra (root certificate) illeszkedik-e. Ha igen, akkor a JAR-t elfogadják az igazolás tulajdonosától származónak. Az aláírás alapján meghatározzák a midlet sorozathoz tartozó engedélyeket. Ezeket az

engedélyeket a telefonba épített policy tárolja. Ez a policy a szokásos implementáció szerint fix, a gyártáskor rögzítődik és nem lehet manipulálni. Tulajdonképpen a policy azt mondja meg, hogy egy bizonyos aláíráshoz milyen engedélyek tartoznak. A midlet sorozatnak emellett meg kell igényelnie az általa használni kívánt engedélyeket a manifest fájlban.

3.3.3.3. MIDP 3.0 - A jövő (ami a jelenben van)

Mivel a fejlődés nem áll meg ráadásul a MIDP 2.0 sem teljesen felel meg a mai elvárásoknak, ezért folyamatosan újabb és újabb csomagokkal bővítik. Az egyes bővítéseket úgynevezett JSR-ek (Java Specification Request) formájában nyújtják be a résztvevő tagok a MIDPEG csoportnak. Ezek a bővítések legtöbbször opcionálisan kerülnek bele a MIDP 2.0-ás eszközökbe, hiszen nem tartoznak a MIDP 2.0 alapkövetelményeihez. Nagy valószínűséggel a MIDP 3.0 (JSR-271)már tartalmazni fogja őket. Néhány érdekesebb bővítés:

- ✔ Wireless Messaging API 1.0 (JSR-120) és Wireless Messaging API 2.0 (JSR-205): SMS és MMS küldések és fogadása MIDlet-ekből
- ✔ Mobile Media API (JSR-135): hang és videó fájlok lejátszása és vezérlése
- ✔ Mobile 3D API (JSR-184): 3D-s grafikus alkalmazások készítése. Többnyire szoftveres renderelő eszközt használva.
- ✔ Bluetooth API (JSR-82): Bluetooth szerver és kliens kapcsolatok létrehozása.
- ✔ PDA Optional Packages (JSR-75): naptár, névjegyzék és az eszköz fájlrendszerének (belső memória, memóriakártya) elérése.

A MIDPEG csoport meghatározott egy olyan eszközigényt amely alapján a MIDP 3.0 minimális követelményei:

- ✔ 176x220 pixel képernyő méret
- ✔ 16-bit színmélység
- ✔ egy-, kétkezes billentyűzet, érintőképernyő esetleg scrollozható kerék
- ✔ 1 MB ROM a CDC vagy CLDC számára
- ✔ 512 kB ROM az alkalmazások számára
- ✔ 1 MB RAM a halomterületnek (Heap)
- ✔ kétirányú, vezetéknélküli, időszakos, korlátozott sávszélességű kapcsolat
- ✔ multimédiás támogatásként hangképzés, hangvisszajátszás

A JTWI kijelöli az utat a Java-telefonok új generációja számára.

A Java Technology for Wireless Industry (JTWI) egy szabványgyűjtemény, ami azt írja le, mit kell támogatnia egy Jáva-képes telefonnak. Ez azt célozza megelőzni, hogy a telefonok sok, egymással nem kompatibilis API-készletet támogassanak. A JTWI-t a JSR-185 írja le és ezeket a főbb megkötéseket tartalmazza.

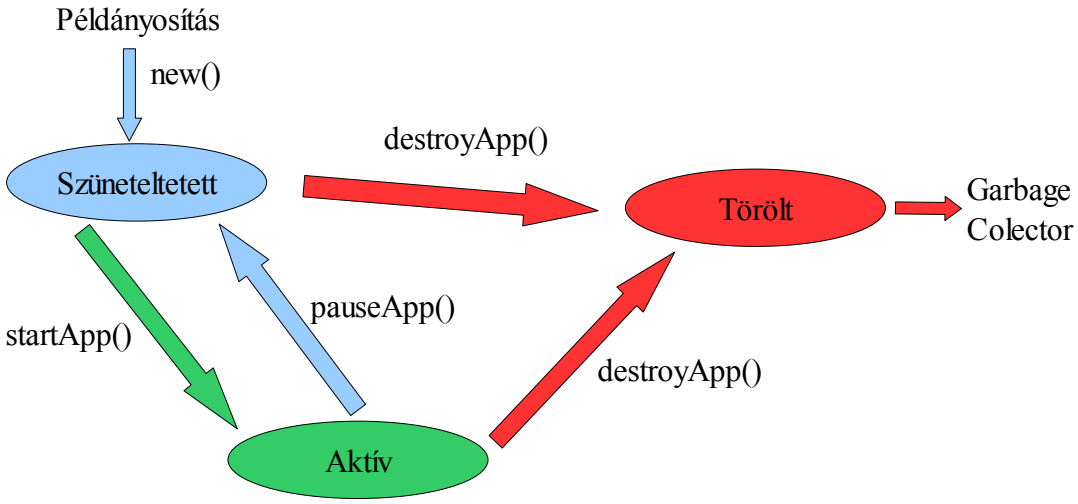
- ✔ Egy JTWI készülék CLDC 1.0 alapú
- ✔ MIDP 2.0 szabványt teljesen támogatja
- ✔ Támogatja a Wireless Messaging API 1.1-et (JSR-120). SMS-ek küldése és fogadása
- ✔ Támogatja a Mobile Media API-t (JSR-135). Ez MIDI lejátszás és kameravezérlést biztosít

3.3.4. MIDlet

MIDlet-eknek nevezzük a MIDP környezetben futó Java alkalmazásokat. Ha a webről egy alkalmazást töltünk le, akkor valójában nem a MIDlet-et töltjük le és indítjuk, hanem egy ún. MIDlet csomagot (suit), ami egy, vagy több MIDlet-et tartalmaz összecsomagolva. A MIDlet suite többnyire több, hasonló funkciót ellátó vagy együttműködő MIDlet összessége. A MIDlet-eket egy AMS (Application Management Software – Alkalmazásvezérlő Rendszer) tölti be, aktiválja, szóval menedzseli. Az egy suite-ban lévő MIDlet-ek osztozhatnak az erőforrásokon (adat, grafika), az ugyanabban lévő MIDlet-ek hozzáférhetnek egy suite-beli MIDlet információkhoz, míg más suite-ban lévő MIDlet-ek erőforrásaihoz nem. A Java ME MIDlet osztályokat használ a programok reprezentációjára.

Egy MIDlet, egy olyan osztály, ami kiterjeszti a javax.microedition.midlet.MIDlet absztrakt osztályt, és implementálja a startApp(), pauseApp() és destroyApp() metódusokat, amik hasonlóak a java.applet.Applet osztály start(), stop(), és destroy() metódusainak működéséhez. Ezen kívül egy MIDlet egyéb közönséges Java osztályokat is tartalmazhat, amiket együtt lefordítva és .jar fájlba (Java Archive Resource) csomagolva kapjuk az úgynevezett MIDlet csomagot. Tovább bővítve a lehetőségeket, egy .jar MIDlet csomag akár több MIDlet-et is tartalmazhat, amik megosztva használhatják egy .jar fájlba csomagolt többi erőforrást, osztályt stb. A .jar fájlról egy .jad leíró fájl (Java Application Descriptor) biztosít a

készüléknek információkat, ahol vannak kötelező illetve lehetnek járulékos információk. Kötelezőek a .jar fájl mérete bájtokban; az URL, ahonnan származik; a MIDlet készítője, verziószáma, a CLDC verziója, a MIDP verziója. A járulékosok pedig a MIDlet-hez rendelt ikon; az adatok tárolásához szükséges memóriaigény; a MIDlet készlet leírása. Egy MIDlet-nek három állapota lehet: active, paused vagy destroyed, azaz aktív, szüneteltetett vagy törölt.



6. ábra: A MIDlet életciklusa

Az aktív állapot azt jelzi, hogy a MIDlet elindult, és fut, azaz a benne implementált funkciók működésben vannak. Ez az állapot a JVM által meghívott startApp() metódus elindulásával veszi kezdetét. Felfüggesztett állapotban (amit a notifyPaused() metódus kezdeményez) minden, az addig a MIDlet-hez társított erőforrás felszabadul, viszont a program készen áll azok azonnali, újbóli lefoglalására. A notifyDestroyed() metódus hatására a MIDlet destroyed állapotba kerül, véglegesen felszabadítja az összes, általa lefoglalt erőforrást, és a GarbageCollector aktiválására vár. Egy MIDlet osztály létrehozásánál komponenseket lehet tenni a telefon grafikus kijelzőjére. Például létrehozhatunk egy form-ot, és arra komponenseket pakolhatunk, majd a form-ot beállítjuk aktuális képernyőnek. Lássuk a klasszikus példán keresztül, hogyan épül fel általánosságban egy nagyon egyszerű Java ME program:

```
//importálás, mert szükségesek
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * @author Buda
 */

//létrehozzuk a Hello nevű MIDlet osztályt
//(a CommandListener osztály az események kezelésére szolgál)
public class Hello extends MIDlet implements CommandListener{
//a Form osztálybeli form adattagunk
private Form form;
//Hello konstruktor
public Hello(){
      //form példány
      form = new Form("Hello MIDlet");
      //statikus szöveg felvitele a formra
      form.append("Ide jön ez a klasszikus szöveg ami meg fog jelenni:");
      form.append("Hello World!");
      //kilépés gomb hozzáadása
      form.addCommand( new Command( "Kilépés", Command.EXIT, 1 ) );

      //CommandListener regisztrálása, hogy a kilépés gomb működjön
      form.setCommandListener( this );
      }
      //a MIDlet indításához szükséges metódus
      public void startApp(){
      //kijelző-hivatkozás lekérése
      Display display = Display.getDisplay(this);
      //aktuális kijelző megadása
      display.setCurrent( form );
      }
      //a pauseApp metódusra most nincs szükség, de implementálni kell
      //mindig, mivel absztrakt metódus
      public void pauseApp() { }
      //a destroyApp metódus törli a form referenciát a MIDlet
      //befejezésekor
      public void destroyApp(boolean unconditional){
      form = null;
```

```
        }
        //A CommandListener interface implementálásához egy metódust kell
        //implementálnunk, ez pedig a commandAction
        public void commandAction(Command c, Displayable d){
        //befejezi a MIDlet futását
        destroyApp(true);
        //értesíti az alkalmazáskezelőt, hogy a MIDlet befejeződött
        //notifyDestroyed();
        }
}
```

4. A projekt, azaz játékfejlesztés játékosan

4.1. A játék leírása

A játék ötletét a Karib-tenger kalózai ihlette, egy nagyon egyszerű kis játék, amelynek lényege, hogy a hajónk, kalózok hajóival találkozik amelyek persze nekiütközhetnek ennek, de a játék célja az, hogy minél több kalózhajót kerüljön ki a mi hajónk.

Egy Nokia S80 emulátor segítségével lehetett tesztelni a játékot, ami egy valódi mobiltelefont szimulál, így nem kellett minden verziót külön telefonra tölteni, ha tesztelni szerettem volna. Ettől függetlenül a kész változat egy Nokia 6120 classic telefonon is ki lett próbálva. Tökéletesen egyforma működést mutatott az emulátor a telefonnal. A játékot a telefon gombjaival lehet irányítani.

4.2. Fejlesztői környezet

4.2.1. NetBeans IDE 6.0.1

A NetBeans IDE olyan fejlesztői környezet, amely lehetővé teszi a programozók számára, hogy programokat írjanak, fordítsanak, teszteljenek, hibakeresést végezzenek az alkalmazásokban, majd profilozzák és telepítsék a programokat. Java nyelven íródott, de bármilyen más programozási nyelvet is támogat. A NetBeans IDE szoftver számos modullal bővíthető. A NetBeans IDE, GPL - General Public License (Általános Nyilvános Licenc) egy általános célú nyílt forráskódú licenc, azaz ingyenes termék, és nincsenek érvényben korlátozások a használatára vonatkozóan. A termék nyílt forráskódú és ingyenesen használható kereskedelmi vagy nem kereskedelmi célokra. A forráskód a CDDL (Common Development and Distribution License) (közös fejlesztési és terjesztési licenc) alapján áll rendelkezésre újrafelhasználásához.

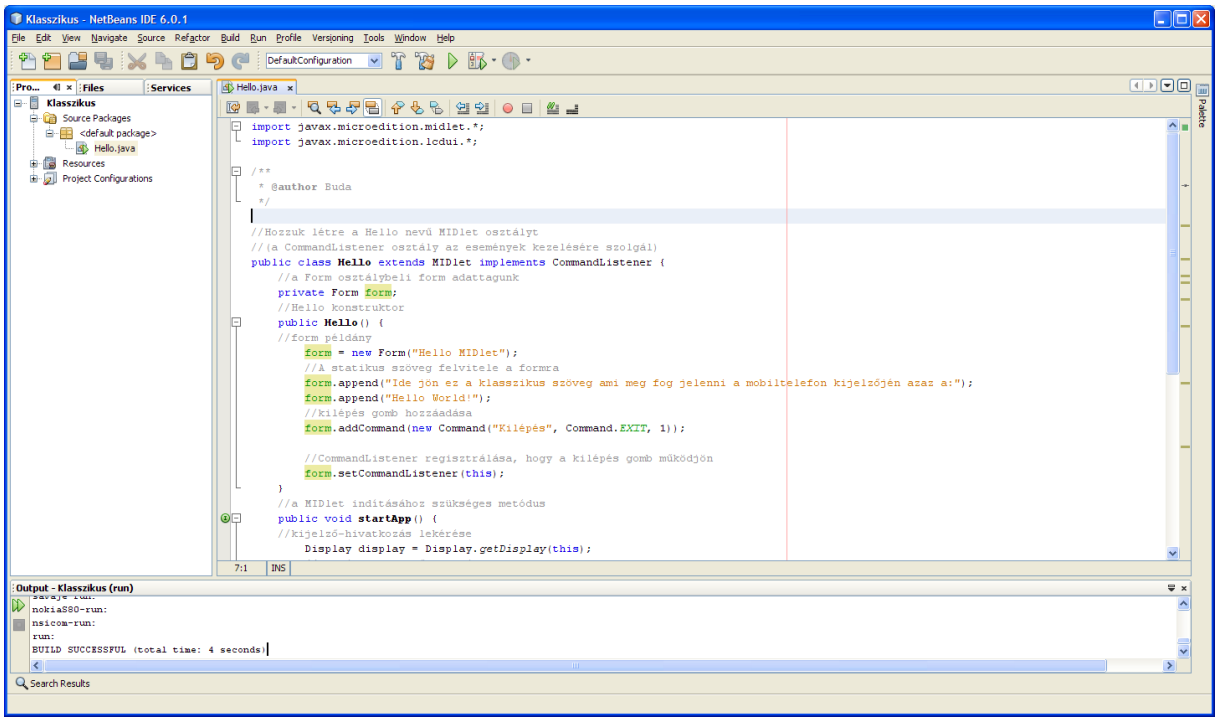
Persze ha fejleszteni akarunk és miért ne akarnánk, akkor fog kelleni még egy Java fejlesztői csomag is, legyen ez a Java SE Development Kit 6, vagy Java futtató környezet

JRE6. Természetesen, ha a JDK-t választottuk, akkor már nincs külön szükség a JRE letöltésére. Mindezektől én eltekinthettem, mert olyan verzióját választottam a NetBeans-nak ami már tartalmazza mindezt.

Miért is választottam ezt? Talán elsődlegesen azért, mert rengeteg olyan elemet tartalmaz, amely a programozó fejlesztési munkáját nagyban segíti.

Többek között:

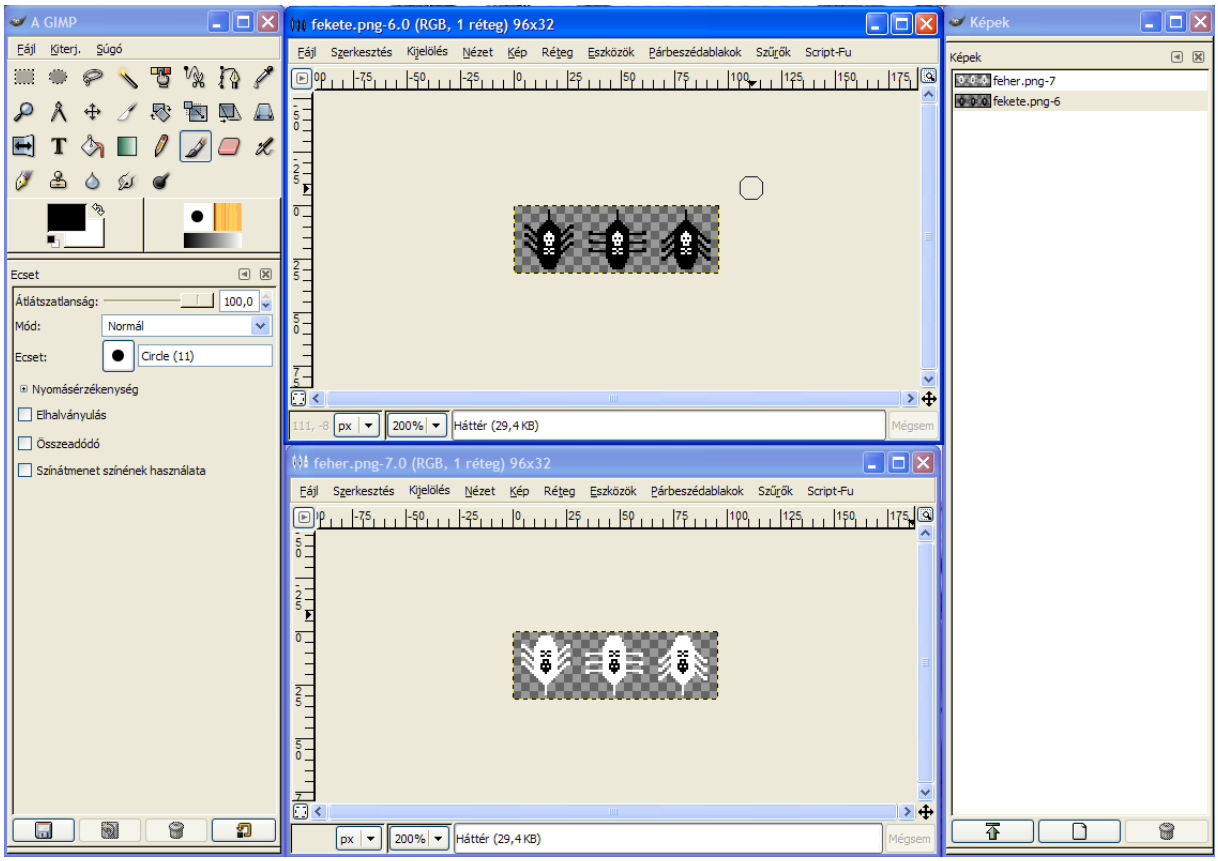
- ✔ Nagyon könnyű és egyszerű projektkezelés.
- ✔ Forráskód kiegészítés, osztályelemek automatikus elhelyezése.
- ✔ Automatikus formázás, színkiemelés.
- ✔ Hibafelismerés és javítás felajánlása már a forrás írása közben.
- ✔ Tesztelés támogatása. Ehhez a NetBeans verzióhoz tartozik még egy Mobility Pack, amiben a Sun Java Wireless Toolkit 2.5.2 verziója is megtalálható, amelyek a CLDC,-MIDP,- és szabványos API-készleteket tartalmazzák és szükségesek a fejlesztéshez. Tehát így egyben letölthető az egész fejlesztői csomag a NetBeans honlapjáról, ellentétben a korábbi verziókkal, ahol a különböző csomagokat külön kellett letölteni és telepíteni.



7. ábra: NetBeans IDE 6.0.1

4.2.2. GIMP 2.2.17

A GIMP - GNU Image Manipulation Program egy multiplatformos képszerkesztő program, amely alkalmas a képretusálástól kezdve a legegyszerűbb képformátum alakításán keresztül a legösszetettebb képmanipulációs eljárások kivitelezéséig. Ennek a programnak a segítségével készítettem a grafikus objektumokat.



8. ábra: A sokrétű képszerkesztő

Először kicsit nehézkesen indult a használata de mint mindenhez, előbb-utóbb hozzáidomul az ember, és nem olyan idegen már a használata sem. Ez egy viszonylag régebbi verzió, és tudom, hogy létezik belőle újabb, amit nagy valószínűséggel minél hamarabb használni is fogok.

4.3. A játék felépítése, elemzése

A Kalózok egy MIDlet-ből (Kaloz_Midlet) és egy Class-ból (Vaszon) osztályból áll. A

Vaszon osztályban van magának a játéknak a működése, míg a MIDlet tulajdonképpen a java programok Main osztálya. Minden mobiltelefonra készített alkalmazás tartalmaz egy MIDlet-et. Az én projektemben ezek a Kaloz csomagban foglalnak helyet, ami nevezetesen így néz ki:

```
package Kaloz;*
```

Nyilvánvalóan ez nem kötelező jellegű, hogy csomagban legyenek, de amennyiben nem tenném csomagba az osztályokat, midlet-eket, akkor egy ún. <default package>-be kerülnek és ha túlságosan nagyra nőne a forrás, akkor nehezebben áttekinthető lenne. A képek pedig amelyeket rajzoltam és a játékban felhasználtam, az Anim nevű csomagban foglalnak helyet. A szükséges csomagokat a MIDlet-hez importáltam, mert szükség van rá, hiszen minden midlet-nek ki kell terjesztenie magát a MIDlet osztályt, ezért kell a javax.microedition.midlet. Az osztály 3 ismerős metódust tartalmaz, ezek a startApp(), pauseApp(), destroyApp(), amelyeket a MIDlet-nek kell implementálni. Az alkalmazásokban létrehozható felhasználói felületek készítésére és vezérlésére tartalmaz osztályokat a javax.microedition.lcdui, ráadásul a programírás is lényegesen leegyszerűsödik. Az importálás így néz ki:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Kaloz_Midlet extends MIDlet implements
CommandListener {
    private Display mobilKijelzo;
    private Command jatekGomb;
    private Command kilepesGomb;
    private Vaszon Kaloz_Vaszon;
```

A létrehozott Kaloz_Midlet után az szerepel, hogy minek a leszármazottja és milyen interfészt implementál, a CommandListener osztály pedig az előbbi példa alapján tudható, hogy az események kezelésére szolgál. A MIDlet minden adattagját és metódusát a Kaloz_Midlet tartalmazza. Ez négy adattagot tartalmaz és ezeket deklarálnom kellett. Az adattagok jelentései:

---

* A forráskódból idézett részleteket világoskék háttérrel jelzem, a NetBeans IDE aktuális sorának színéből merítve az ötletet, bár az világosabb (itt sajnos nem találtam ettől világosabbat)

- ✔ Display mobilKijelzo - Olyan input eszközkezelést reprezentáló osztály, aminek a segítségével lehet a képernyőt beállítani és a készülék egyéb funkcióit bekapcsolni
- ✔ Command jatekGomb - Kijelzőn megjelenő eseménykezelés, a játék indítása
- ✔ Command kilepesGommb - Kijelzőn megjelenő eseménykezelés, a játék befejezése
- ✔ Vaszon Kaloz_Vaszon - Az osztály, amely a játékot megvalósítja

Most az osztály konstruktorán a sor. Mivel a játéknak valahol meg kell jelenni, tehát elég vizuális ez a téma ahhoz, hogy ne csak a fantáziánkban képzeljük el, ehhez pedig át kell adni a mobilKijelzo-nek a mobiltelefon kijelzőjét reprezentáló objektumot.

9. ábra: A szimulátor

Erre való a this, amit a Dispaly.getDisplay metódussal lehet elérni. Ezután inicializáltam a Command típusú objektumokat, amelyek magasszintű események tulajdonságait tartalmazzák. Ennek az objektumnak a paraméterei sorrendben a következőek: mit fog kiváltani a parancs, azaz mi jelenik meg a kijelzőn, a hozzárendelt parancstípus, végül a prioritás amely egy szám, és annál fontosabb ez, minél kisebb az értéke.

```
mobilKijelzo = Display.getDisplay(this);
jatekGomb = new Command("Kezdés",Command.ITEM, 10);
kilepesGomb = new Command("Kilépés",Command.EXIT, 10);
```

Korábban volt róla szó, hogy egy MIDlet életciklusában milyen folyamatok zajlódhatnak le. Így van ez a Kaloz_Midlet esetében is, melyet örökölt az importált csomagból, csak egy kicsit a Kalózok igényeihez képest újraimplementálva így néz ki a startApp() metódus:

```
public void startApp(){
    Kaloz_Vaszon = new Vaszon();
    mobilKijelzo.setCurrent(Kaloz_Vaszon);
    Kaloz_Vaszon.addCommand(jatekGomb);
    Kaloz_Vaszon.addCommand(kilepesGomb);
    Kaloz_Vaszon.setCommandListener(this);
    }
```

A kötelezően implementálandó metódusban először is példányosítom a saját Vaszon osztályomat, és a midlet képernyőjét a Kaloz_Vaszon-ra állítom. A Kaloz_Vaszon-hoz hozzárendelem az addCommand() utasítással, az előbb inicializált parancsokat és végül a parancsfigyelőt – setCommandListener – beállítom. A commandAction() metódus implementációja, amely szükséges a CommandListener interfészhez, biztosítja azt, hogy reagáljon arra az eseményre, amennyiben a „Gombokkal” történne valami. Egész pontosan arra vonatkozik ez a megállapítás, hogy a midlet-nek le kell kezelnie azt az eseményt a commandAction viselkedésével, ha a parancsok használatba kerülnek.

Jöjjenek azok a MIDlet életének ciklusbeli metódusai melyek ugyan kötelezőek, hiszen a MIDlet-ből öröklődtek tehát kihagyni nem lehet őket, viszont tartalommal nem töltöttem fel, mivel a programban most nincs rájuk szükségem. Ez a szüneteltetés metódusa,

```
public void pauseApp() {}
```

...valamint a törlés metódusa:

```
public void destroyApp(boolean unconditional){}
```

Befejezésül pedig annak a részletezése jön, hogy mi történik akkor, ha egy esemény bekövetkezik, azaz annak commandAction() metódusnak a kifejtése, amiről az imént beszéltem. Ezekkel a parancsok használatakor végrehajtódó eseményeket lehet beállítani.

```
public void commandAction(Command command,Displayable
 displayable){
      if (command == jatekGomb){
            indulas();
      }
      if (command == kilepesGomb)
            befejezes();
      }
```

Nyilvánvaló, hogy akkor ezeket a metódusokat ki is kell fejtsem, hiszen hivatkoztam rájuk. A jatekGomb parancs végrehajtásakor lefutó indulás() metódus pl. eltünteti a kijelzőről a Játék gombot és láthatóvá teszi a kijelzőn a „grafikus” képernyőt.

```
public void indulas(){
      Kaloz_Vaszon.removeCommand(jatekGomb);
      Kaloz_Vaszon.betoltes();
      }
```

A befejez() metódus implementálásakor a játék leállításához szükséges feltételeket állítom be. A Kaloz_Vaszon betolt adattagját false-ra állítom, mivel nem lesz tovább szükségem a kijelzőre, ezért azt null-ra változtatom a megjelenítendő objektumot, a destroyApp() true paramétere a midlet futását mindenképpen befejezi és felszabadítja a lefoglalt memóriaterületeket, valamint a notifyDestroyed() jelzi az alkalmazásvezérlőnek, hogy a midlet befejezett állapotba került és lezárhatja azt.

```
public void befejezes() {
      Kaloz_Vaszon.betolt = false;
      mobilKijelzo.setCurrent(null);
      destroyApp(true);
      notifyDestroyed();
      }
```

Tehát így néz ki valahogy az én main osztályom, hiszen eddig tulajdonképpen a midlet-emet írtam körbe.

Amikor a játékot elindítom akkor egy viszonylag egyszerű képernyőt lehet látni hiszen a háttérben egy kép látszik, azon egy felirat található, a grafikus rész alatt pedig két parancs, a Kezdés és a Kilépés fedezhető fel. Kivitelezhető lenne akár komplett menürendszert előállítani parancsok segítségével. A mobilos alkalmazások felhasználói felülete nagyban különbözik az asztali gépek felhasználói felületétől mármint a kezelésüket tekintve. Lényeges szempont, hogy egyszerűek, illetve egyértelműek legyenek az informatikában nem jártasak számára is. Könnyen történjen a mobiltelefonok billentyűzetét használva a navigáció. A legszembetűnőbb dolog az, hogy a mobilok nem rendelkeznek egérrel mint az asztali gépek, vagy ceruzával mint a PDA-k. Sokkal kisebb a kijelzőjük mint ahogyan megszokhattuk azt az asztalnál ülve egy monitor előtt. Mindezeket a szempontokat figyelembe véve kell megtervezni a mobiltelefonra készülő alkalmazásokat. Erre egyébként számtalan lehetőséget biztosít a fejlesztői környezeten keresztül maga a technológia.

A grafikus felhasználói interfész a kisméretű kijelzőre, illetve a bemenetet megvalósító és más natív eljárásokra méretezett. A MIDP a telefon billentyűzetének és egyéb gombok teljes kihasználásával biztosítja az intuitív navigációt és adatbevitelt. A játékosztály megint csak az általam előállított csomaghivatkozással, valamint a szükséges csomagok importálásával kezdődik. Csakhogy ebben található egy olyan API-készlet amely jelentősen meghatározza a mobilra készített játékok fejlesztését. Ebben a fejlesztő rendelkezésére állnak olyan osztályok és azokban olyan metódusok, amelyekkel viszonylag rövid idő alatt rendkívül egyszerűen lehet mobilos játékokat előállítani. Ez a MIDP 2.0-ban jelent meg és segítségével animált játékok is készíthetőek.

```
package Kaloz;
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
```

Ezeket annyira lényegesnek tartottam, hogy részletesebben is leírom, milyen fontos osztályok találhatóak bennük, melyeket én is alkalmaztam.

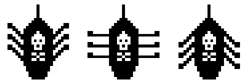
A *javax.microedition.lcdui.game* csomag használt osztályai:

- **GameCanvas** – a játék felületének alapját adó osztály. Egyenletesen mozgó animációt eredményez (Dupla Puffer technika). A dupla-pufferelés azt jelenti, hogy két lapom van. Egy amit a felhasználó lát, és egy, amin dolgozhatok. A kettőt egy művelettel kicserélhetem. Így a sok időt emésztő rajzolást a háttérben végezhetem, és csak akkor mutatom meg a felhasználónak a képet, ha az elkészült. Ha nem használnám ezt a módszert, az animációm biztosan villogna, ami nem túl szép. A pufferkezelő stratégia objektumával tudom a háttérlaphoz tartozó Graphics objektumot lekérdezni, illetve a cserét végrehajtani.

- **Layer** – a játék egy vizuális elemét képviselő absztrakt osztály. Úgy kell elképzelni, mint egy grafikus programban a réteget, illetve annak a kezelését. A leszármazottai a Sprite és a TiledLayer.

- **Sprite** – egy olyan alapvető és az egyik legfontosabb vizuális elem, aminek a segítségével lehet animációt létrehozni. Egy kép tartalmazza az animáció képkockáit. Ezek a képkockák váltakoznak a képernyő frissítései során. Minden egyes képkocka azonos méretű, és 0-tól kezdődően egy sorszám van hozzárendelve, ez a képkockák száma-1-ig tart. A program futása során ezekkel a számokkal lehet hivatkozni az adott képkockára, ami alapértelmezés szerint sorrendben történik, de ettől el lehet térni. Nálam ez a pl. a hajó(k) mozgásánál nyilvánul meg a legkézenfekvőbben. Persze hasonlóképpen működik a fehér hajók mozgása is, azért használtam a többes számot. Az evezők mozgása így van szimulálva az alábbi módon:

  ```
  fekete.setFrameSequence(new int[]{0, 1, 2, 1, 0});
  ```



10. ábra: Evezés

Az azonos mozzanatokat ismétlésekkel lehet megoldani. Tehát a képkockákat a Sprite objektum képelemének feldarabolásával kapom meg. Ezt a darabolást, – ha úgy akarja a fejlesztő – többféleképpen is meg lehet tenni nemcsak vízszintesen, hanem akár függőlegesen vagy a kettőt összekombinálva is.

- ✔ **TiledLayer** – a másik vizuális elem, ami celláknak olyan rácshálójából tevődik össze, amivel „ki lehet csempézni" akár nagyobb területeket is egy viszonylag kis képpel vagy képekkel. Általában olyan képeknél használatos, ahol sok a képkockák ismétlődése. A kalózoknál a csempézett háttér, és a háttér szerkezetéhez szükséges tömb megadása után, maga a tömb szerkezete valamint annak feltöltése ciklusok segítségével nagyon egyszerűen történt:

```
TiledLayer hatter;
int [][] palya;
palya = new int [][]{
     {1, 2, 6, 6, 3, 4},
     {5, 6, 6, 2, 7, 8},
     {9, 10, 6, 6, 11, 12},
     {13, 14, 6, 6, 15, 16},
     {17, 18, 6, 6, 19, 20},
     {1, 2, 6, 6, 3, 4},
     {5, 6, 6, 2, 7, 8},
     {9, 10, 6, 6, 11, 12},
     {13, 14, 6, 6, 15, 16},
     {17, 18, 6, 6, 19, 20},
     {1, 2, 6, 6, 3, 4},
     {5, 6, 6, 2, 7, 8},
     {9, 10, 6, 6, 11, 12},
     {13, 14, 6, 6, 15, 16},
     {17, 18, 6, 6, 19, 20},};
```

```
hatter = new TiledLayer(
        palya[0].length, palya.length, csempek, 44, 44);
        for(int i=0; i<palya.length;++i)
                for(int j=0; j<palya[0].length;++j)
                        hatter.setCell(j, i, palya[i][j]);
```

- ✔ **LayerManager** – az az osztály, ami összefogja és kezeli a Layer-ket. A LayerManager rajzoló metódusának a segítségével minden olyan Layer kirajzolásra kerül, amely az adott LayerManager-hez tartozik. Az alapértelmezés szerinti hozzáadási sorrendet meg lehet változtatni.

  Amennyiben ezeket siker koronázza, akkor kirajzolódik előttem a kijelző hátttereként egy „hömpölygő” folyó és elindulhatok becserkészni a leendő „áldozatomat” valahogy így:

11. ábra: Portyázás

A `javax.microedition.lcdui` csomag használt osztályai:

- **Image** – az osztályt arra használom, hogy grafikus képadatokat tölthetek be a midlet-be. Ennek segítségével lehet képeket készíteni alkalmazáshoz. Az állóképeken létrehozás után már nem lehet változtatni. Egy mozgóképből készíthető állókép a createImage() metódushívással. A képobjektum PNG (Portable Network Graphics) formátumú lehet, amelyet a W3C fejlesztett ki olyan célból, hogy egy olyan tömörítési eljárást vezessen be amely nem áll jogvédelem alatt. Ahhoz, hogy a képet meg lehessen jeleníteni a mobiltelefonon, annak bizonyos előfeltételei vannak. Ezek közé tartozik, hogy a PNG fájl és a képobjektum hosszának illetve szélességének meg kell egyeznie. Ha túl nagyok az értékek, akkor memóriakapacitás hiányában nem biztos, hogy megjeleníthetőek, minden színt használni lehessen, bár ez nagymértékben függ a készüléktől. A Sprite-ok kereteit tartalmazó képek és az őket reprezentáló objektumok:

```
Image kezdokep, kep, kep2, bumm;
Sprite kezdo, fekete, feher, robban;
```

Aztán a Vaszon konstruktora, ahol hivatkozok ezekre az objektumokra:

```
public Vaszon() {
    super(true);
    try {
        kezdokep = Image.createImage("/Kalozok_kezdo.png");
        kep2 = Image.createImage("/fekete.png");
        kep = Image.createImage("/feher.png");
        bumm = Image.createImage("/bumm.png");}
    catch(java.io.IOException e) {
        System.out.println("Hiányzik valamelyik objektum!");
    }
    void betoltes(){
        new Thread(this).start();
    }
```

A képek létrehozását try catch blokkba írtam, mivel itt kiváltódhat kivétel abban az esetben, ha valamilyen oknál fogva a képek nem tölthetőek be. Az animáció mozgás, amit egy külön szál, a példányosított Thread() osztály vezérel, amelynek

paraméterül adtam magát az osztályomat. Ezzel létrehoztam egy szálat, mely szálon a játék majd futni fog. A start() metódus pedig indítja a programszálat.

- **Graphics** – lehetővé teszi a kétdimenziós grafikai műveleteket megvalósítását a kijelzőn. Lehet rajzolni síkidomokat, vonalakat és ha a készülék támogatja a megfelelő színmélységet, akkor ki is lehet színezni őket. A kijelző alapértelmezett koordináta rendszere a bal felső saroktól kezdődik (0,0 koordináta). Az x tengely balról jobbra, az y pedig felülről lefelé halad. A képet vagy esetleg rajzot, úgynevezett referenciapontokon keresztül tudom „megfogni” és a meghatározott helyére illeszteni. A játék kezdő képernyőjének a megjelenítése ily módon történik:

```
Graphics g = getGraphics();
        g.drawImage(kezdokep, getWidth()/2, getHeight()/2,
        Graphics.VCENTER | Graphics.HCENTER);
        }
```

… és ezt látom

12. ábra: Kezdőképernyő

- ✔ **Font-** a betűtípus kezelésére szolgáló osztály, amely a fontokat és azok tulajdonságait reprezentálja. Beállítások segítségével kombinálható össze a nekem megfelelő betű megjelenése, stílusa, mérete.

A lényegi rész tulajdonképpen egy ciklusban történik, hiszen amíg az fut, addig tart a küldetés, tehát maga a játék. Mikor ez elindult, folyamatosan vizsgálni kell, hogy lenyomták-e a telefon gombjait, mert ennek alapján végez az általam vezetett hajó meghatározott mozgásokat. Nemcsak jobb és bal irányban tud mozogni, hanem föl-, és lefelé is a folyón. Amennyiben esetleg túlhaladtam a kijelzőn, mondjuk a tetején, akkor sincs semmi gond, mert ugyanilyen ellenkező irányú navigálással szépen visszatér a megfelelő, „látható” részre. Közben folyamatosan folyik a folyó, tehát azok a hajók amelyeket nekem kerülgetnem kell egyre, s másra tűnnek fel a mobil kijelzőjén nyilván ezeket újra és újra ki kell rajzolni.

```
while(betolt) {
    int lenyomottBillenytu = getKeyStates();
    if((lenyomottBillenytu&GameCanvas.RIGHT_PRESSED) != 0) {
        if(jatekosOszlop+3 < getWidth() + fekete.getWidth())
            jatekosOszlop += 3;
    }
    else if((lenyomottBillenytu&GameCanvas.LEFT_PRESSED)!=0) {
        if(jatekosOszlop-3 > - fekete.getWidth())
            jatekosOszlop -= 3;
    }else if((lenyomottBillenytu&GameCanvas.UP_PRESSED)!= 0){
        feketeSor -= 3;
    }else if((lenyomottBillenytu&GameCanvas.DOWN_PRESSED)!= 0)
        feketeSor += 3;
    }
    g.setColor(0xffffff);
    g.fillRect(0, 0, getWidth(), getHeight());
    if(hatterIndex < 0)
        ++hatterIndex;
    else hatterIndex =-(palya.length * 44 - getHeight());
    hatter.setPosition(0, hatterIndex);
    hatter.paint(g);
    ...
```

13. ábra: Támadás

Számomra az ütközések érzékelése, lekezelése jelentette a legnagyobb kihívást, de amikor felfedeztem, hogy milyen lehetőségeim vannak, nincs szükség bonyolult, akár egyenként kivitelezendő koordináta számításokat végezni és nem kell bonyolult algoritmusokkal az időt tölteni, akkor tartottam csak igazán fontosnak, hogy így belemélyedtem a Java-ba. Ezek helyett csak egy egyszerű collidesWith() metódust kell meghívni, amely true értékkel tér vissza, ha az első paraméterében megadott Sprite, ütközik a második paraméterében megadottal. Amit figyelembe kell venni, nem más mint a Sprite-oknál a referenciapixel. Íme a számomra rendkívül egyszerűen kivitelezhető részlet:

```
if ((fekete.collidesWith(feher1, true)) ||
(fekete.collidesWith(feher2, true)) ||
(fekete.collidesWith(feher3, true))){
      robban.setRefPixelPosition(fekete.getRefPixelX(),
      fekete.getRefPixelY()-10);
```

```
        for (int i=0;i<3;i++){
              robban.nextFrame();
              robban.paint(g);
              flushGraphics();
              try{Thread.sleep(50);}
              catch(InterruptedException e) {}
        }
        befejez = true;
}
```

… és annak hatására megjelenő kép a mobilkijelzőn, ami egyben a játék befejezését is jelenti.

14. ábra: Nem sikerült

5. Összefoglalás

A dolgozatom célja az volt, hogy tapasztalatokat szerezzek egy új területen, és készítsek egy játékot mobiltelefonra, amely a nyelv fontosabb oldalait bemutatja. Ezeket úgy érzem sikerült megvalósítani. Önálló munkával elkészítettem egy játékot J2ME nyelven, ami teljesíti az elvárásokat, tartalmaz grafikát, nyelvi elemeket, emulátoron és mobilkészüléken egyaránt használható. Eközben ismereteket szereztem a Java-ról, és egy játék elkészítésének általános módszeréről.

A program jelenlegi állapotában alkalmas lehetne játékállások elmentésére és visszatöltésére, sőt az elkészített játék könnyen továbbfejleszthető más irányokba is, melyek javíthatják a játék élményét (színek finomítása, több objektum használata, stb.). Ahogy az elemzésből kiderült a MIDP csomagok jelentősen megkönnyítették a fejlesztést, ha például a Sprite-ok, vagy a Layer-ek kezelésére gondolok. Ami kimondottan tetszett nekem a játékom megvalósításánál az, hogy a játékfelületen két tárgy összeütközését sokkal könnyebben meg lehet vizsgálni, mint más körülmények között. Sok számolgatástól megkíméli a programozót, ugyanakkor széles skála nyílik egy fejlesztő számára. Úgy gondolom, hogy sikerült bemutassak pár lehetőséget ebből. A továbbfejleszthetőség, persze mindig ott van, mint előre kitűzött cél.

A pálya vonalvezetésébe még nehezítést belerakni, az ütközések számához eseményeket hozzárendelni, a partról támadásokat végezni, ugyanakkor a hajót menet közben megszerezhető pontokkal „javítani” stb. De ebben a kis játékban is benne vannak a mobiltelefonos játék fejlesztésének apró trükkjei, fortélyai. Ugyanakkor a telefonoknak nagyon sok funkciója van a játékon kívül, amihez természetesen más ismeretek is szükségesek. Számtalan lehetőség van, amit a mai okostelefonok már képesek megvalósítani. Most engem viszont a játék elkészítése „hozott lázba”, hiszen azt tartom fontosnak, hogy érdekeljen az, amit csinálok. Mivel a játékok elég közel állnak hozzám, így számomra egyértelmű, hogy egy olyan dologgal kezdjem a mobilok működésének elsajátítását, ami érdekel is.

6. Irodalomjegyzék

A szakdolgozat írásához felhasznált könyvek, jegyzetek, cikkek, statisztikák és internetcímek gyűjteménye

Könyvek, jegyzetek

- ✔ Bátfai Norbert - Nehogy már a mobilod nyomkodjon Téged! (Egyetemi Könyvtár 2007)
- ✔ Nagy Gusztáv - Java programozás (GANF Jegyzet 2007)
- ✔ Angster Erzsébet - Objektumorientált tervezés és programozás JAVA (4 Kör Bt. 2001)
- ✔ Vartan Piroumian - Wireless J2ME Platform Programming (Sun Microsystems Press 2002)
- ✔ Andrew S. Tanenbaum - Számítógép-hálózatok (Panem 2004)

Internet, cikkek, statisztikák

- ✔ A mobil eredete - http://www.mobilport.hu/?r=7814
- ✔ NHH gyorsjelentés a mobilhasználatról http://www.nhh.org.hu/index.php?id=hir&cid=5591
- ✔ Az adat a biznisz - http://www.piacesprofit.hu/?s=32&mr=1986&p
- ✔ Európa a kártyás mellett voksol - http://www.sg.hu/cikkek/62974/europaban_a_kartyas_mobil_a_nyero
- ✔ Legfrissebb hírek - http://www.sg.hu/cikkek/63481/tovabb_csokkenti_a_mobilvegzodtetesi_dijakat_az_nhh
- ✔ Java mint programozási nyelv - http://hu.wikipedia.org/wiki/Java_(programozási_nyelv)
- ✔ Java programozás - http://hu.wikibooks.org/wiki/Java_Programozás
- ✔ Java specifikációk - http://jcp.org/en/jsr/overview
- ✔ Java ME Technology - http://java.sun.com/javame/technology/index.jsp
- ✔ Java ME Technology API Documentation - http://java.sun.com/javame/reference/apis.jsp#api
- ✔ NetBeans 6.0.1 - http://download.netbeans.org/netbeans/6.0/final/
- ✔ GIMP a Weben - http://www.gimp.org/

7. Függelék

Az általam készített játékhoz tartozó Kalozok.jad tartalma:

MIDlet-1: Kaloz_Midlet,/ikon.png,Kaloz_Midlet
MIDlet-Data-Size: 14
MIDlet-Jar-Size: 32361
MIDlet-Jar-URL: Kalozok.jar
MIDlet-Name: Kalozok
MIDlet-Vendor: Buda
MIDlet-Version: 2.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0

… és hogy még hitelesebb legyen, egy kép a működő mobiltelefonról egy még korábbi Kalózok 1.0 verzió korában...

15. ábra: Nokia 6120 Classic

8. Köszönetnyilvánítás

Szeretnék köszönetet mondani elsősorban tanáromnak és mentoromnak - aki egyben a konzulensem - Bátfai Norbertnek, aki először is felkeltette érdeklődésemet a Java és ezen belül a mobilos alkalmazások lehetőségének irányában, valamint megkedveltette és kellő vehemenciával próbálta kielégíteni információéhségemet ezen a területen.

A családomnak azért, hogy időről-időre elviselték az általam elviselhetetlennek tűnő megpróbáltatásokat és segítettek minden kis apró figyelmes mozzanattal, ami megújuló és friss erőt adott ahhoz, hogy elkészíthessem alkotásom.

Ezenkívül természetesen a Sun-nak a Java megalkotásáért, valamint a NetBeans-projekt elindításáért és fejlesztéséért aminek a segítségével tudtam „szárnyaimat bontogatni”.