SZAKDOLGOZAT

Olajos Mihály

Debrecen

2008

Debreceni Egyetem

Informatika Kar

Mobil alkalmazás fejlesztése

Témavezető:

Bátfai Norbert

Egyetemi tanársegéd

Készítette:

Olajos Mihály

Programtervező Informatikus

Debrecen

2008

Tartalomjegyzék

1. Bevezetés

1.1 Miért pont mobil alkalmazások fejlesztése?

Már több éve rendelkezem mobiltelefonnal. Kezdetben csak telefonáltam, SMS-t írtam és játszottam vele. Azóta szerves része lett a mindennapi életemnek. Ami azt illeti többször megfordult a fejemben, hogy jó lenne saját játékot vagy alkalmazást készíteni rá és mindig visszatérő gondolataim voltak ezzel kapcsolatban. És ekkor még csak 2000-et írtunk. Ebben az időben még fogalmam se volt, hogy hogyan kezdhetném el. Mindaddig amíg (2007-ben) meg nem megismerkedtem a Java nyelvvel.

Ezért gondoltam azt, hogy szakdolgozatommal megmutassam milyen lehetőségeink vannak, ha mobil eszközökre szeretnénk alkalmazás(oka)t készíteni és továbbfejleszteni.

Alkalmazások alatt értem a játékokat, zene és videó lejátszókat, az üzleti életben használt programokat és minden olyan programot amit az asztali PC-ről a mobilkészülékre át tudunk írni.

Tanulmányaim során különböző programozási nyelveket ismertem meg, melyeken keresztül láthattam és tapasztalhattam a programozásban és az alkalmazások fejlesztésében levő lehetőségeket. Egyik ilyen lehetőség volt amikor életre keltettem az első `Midlet`-emet. Sikerélményem volt – ugyanis alapvetően vizuális beállítottságú vagyok- és azóta foglalkoztat ez a téma. És persze, mint mindenkinek akinek sikerélménye van, én nekem is szárnyaltak a gondolataim a témával kapcsolatban. Ugyanakkor ne gondolja senki, hogy csak sikerélményeim voltak, ehhez hozzátartozik a sok megpróbáltatás, a kisebb nagyobb kudarcok elviselése is. Magamat ismerve egy kisgyerekhez tudnám hasonlítani aki ha elesik, fel áll és megy tovább. Egy kis kitartással és szorgalommal azonban minden problémát megoldhatunk. Nincs is annál felemelőbb érzés amikor egy probléma megoldását saját magunkénak tudhatunk. Megragadott ez a vizuális élmény és az hogy milyen logikusan, jól strukturáltan lehet felépíteni például egy általunk elgondolt, jól megtervezett programot. Az első MIDlet-em előtt csak elméletben volt ismeretem a Java programozási nyelvről. Ettől kezdve persze átment gyakorlati munkába és elkezdődött a tapasztalatszerzés is. Az első találkozásom a Java-val kicsit rémisztő volt, de meg kellet barátkoznom a gondolattal, hogy ha mobil eszközökre szeretnék alkalmazásokat készíteni, akkor jelenleg nincs választásom. Rémisztő

volt mert, egy újfajta gondolkodásmódot (objektumorientáltság), szemléletet (osztályok, objektumok) kívánt meg szemben a hagyományos programozással (pl.:C). Most belegondolva nem bántam meg.

A szakdolgozatommal a következő gondolatmenetet próbálom megvalósítani.

2. fejezet: A bevezetőt követően a saját tapasztalatomat felhasználva ismertetem a múltban elkezdődött és jelenleg is tartó technikai fejlődést az én szemléletemen keresztül.

3.fejezet: Ezután megnézzük a mobilszoftverek helyzetét. Ismertetésre kerülnek a különböző mobil szoftver platformok.

4.fejezet: Ezt követően bepillantunk a Java Technológiába.

5.fejezet: Említésre kerül a Java nyelv, de csak annyira, amennyire a programomban jelenik meg. Tehát a leírás nem lesz teljes körű.

6.fejezet: Szó esik a szoftvertervezésről. A készülékek specifikációt figyelembe véve az alkalmazások tervezése.

7.fejezet: Fejlesztőkörnyezetek és a hozzájuk kapcsolható programok ismertetése. Alkalmazások létrehozásához szükséges programok.

8.fejezet: A programom ismertetése a J2ME-én keresztül.

9. fejezet: Összefoglalás

10. fejezet: Ismertetésre kerülnek még az általam használt rövidítések szakszavak.

11. fejezet: És legvégül ismertetésre kerülnek mindazok a könyvek, cikkek, címek melyek alapot szolgáltattak a szakdolgozatom elkészítésében.

2. Múlt és Jelen

Mint a szakdolgozatom címe is mutatja a most ismertetésre kerülő téma nagyon is aktuális lehet a rohanó világunkban. Olyan dinamikusan, olyan gyorsan fejlődnek a technológiai dolgok a számítástechnika világában, és a mobiltelefonok világában is, hogy amit ma magunkévá teszünk(megtanulunk és használunk), az holnap már a múlté. Ugyanakkor a technológiával párhuzamosan a mobilkészülékek is rohamosan fejlődtek. Emlékszem még arra az időszakra amikor „aktatáskás” telefonnal rendelkezett a cégünk (Kb:1993-94), és pusztán csak telefonálásra volt használható. Nem voltak rajta alkalmazások, játékok, nem volt kijelzője, s nem csenget olyan szépen mint manapság a mai készülékek, a zenelejátszásra

pedig gondolni se tudtunk. A technológia akkoriban nem tette lehetővé a széles körben elterjedt programfejlesztést. A felhasználók körét tekintve, ha azt nézzük, csak a társadalom egy szűk rétegének volt kiváltsága, melyet egyfajta szimbólumként lehet elképzelni. Manapság pedig „cigarettadoboz" méretű és ing vagy farzsebben is elfér e csodás készülék. A mai mobiltelefonok tudását tekintve messzemenően meghaladják a kezdeti készülékeknek és ami a legfontosabb a társadalom minden rétegének elérhetővé vált. A jól megszokott telefonáláson túlmenően, igazi multimédiás készülék vált belőle. Elterjedéséből következően a programfejlesztés is utat tört magának, s jelenleg is több kisebb-nagyobb cég foglalkozik programfejlesztéssel. És hogy mi történt a múlt és a jelen helyzet között? A technika oldaláról tekintve talán nem is célszerű két szélsőséges állapotot megállapítani hiszen, ami biztos az a múlt, ahonnan keletkezett, ahonnan fejlődött, aminek tudjuk az előzményeit és történetét. Másik véglet pedig nincs, mert nem áll meg itt (a mai napon) a technológia hanem rohamosan fejlődik, napról-napra, amelynek részesei vagyunk mi is. És hogy mit hoz a jövő, azt csak találgathatjuk, jobb esetben sejthetünk valamit. Miért mondom, hogy sejthetünk valamit? Hát azért, mert talán mi magunk vagyunk azok akik valami újat, valami merőben másat alkotunk, valami pluszt tudunk nyújtani, esetleg részese vagyunk a technológiai fejlődésnek. Napjaink mobiltelefon szokásait tekintve igen sokrétűek vagyunk, minden ember máshogyan és másra használja a készülékét. Ha ebből a szemszögből nézzük a dolgot akkor viszont tényleg meg lehet határozni két szélsőséges esetet, amikor is az egyik felhasználónak elég, ha olyan készüléke van amin elérhető és ő elérhet bárkit. A technológiai dolgokról nem tud illetve nem is akar tudomást venni. Gondolok most az idősebb korosztályra. A másik felhasználó típus - a szélsőséges eseteket tekintve - pedig az aki a készüléket nem csak telefonálásra, hanem különböző alkalmazások futtatására is használja. Értem ez alatt a zeneletöltést és hallgatást, SMS és MMS szolgáltatások igénybevételét, a játékokat és még sorolhatnám sokáig a mobiltelefonokon futtatható alkalmazásokat. Bár ezek nagyon szélsőséges esetek mégis e két határ között bizonyos szempontok szerint kategorizálhatjuk a felhasználókat. Csoportokat képezhetünk belőlük pl.: életkor szerint. Csoportosítás után jobban meghatározhatjuk, hogy milyen alkalmazásokat igényelhetnek és annak megfelelően készíthetünk programokat.

3. Mobilszoftverek világa

A PC-k világában mindenki szabadon választhatja meg, hogy milyen operációs rendszert telepít a gépére. A kézi eszközök esetében nincs ilyen szabadságunk, az operációs rendszer gyárilag telepítve van, és nincs rá lehetőség, hogy egy gyártó ugyanazt a modellt többféle platformváltozattal is piacra dobja.

Alkalmazásokat különböző platformra és készülékekre készíthetünk, főként mobil eszközökhöz (mobiltelefon, PDA stb.). Az egyik legelterjedtebb, a legtöbb felhasználót érintő készülék a mobiltelefon. Ma már kevés azoknak a száma akik még nem rendelkezik vele. A kezdeti igényeket felülmúlva - amikor még nem volt szó pl.: erőforrás kezelésről - jelentek meg olyan alkalmazások, amelyek a PC-khez hasonlóan szoftver platformot igényeltek.

Havonta jelennek meg az újabb és újabb készülékek, melyek között egyre több az olyan, ami rendelkezik mobiltelefonokra szánt operációs rendszerrel. A szoftver platformok hihetetlen fejlődésen és ütemben mennek keresztül. A jelenleg elérhető platformok a következőek:

- Symbian OS
- Windows CE (Windows Mobile)
- Palm OS
- Android stb.

Ha egy szoftverplatform nyílt akkor az azt jelenti, hogy bárki írhat alkalmazásokat a rendelkezésre álló fejlesztői szoftverekkel a maga elképzeléseinek megfelelően. Természetesen a licenc feltételeket elfogadva és betartva.

3.1. Symbian OS

Talán nem túlzás kijelenteni, de a Symbian piacvezető a mobilkészülékre gyártott operációs rendszerek terén. A Nokia saját kezelőfelületet fejlesztett készülékei számára, melyet több cég is tovább licencelt.(LG, Samsung, Motorola, Siemens stb.) A Nokia kezelőfelületei a S40, S60, S80 és S90 néven ismertek. Ezek közül a legismertebb a S60-as mely más gyártók készülékeiben is megtalálható. Az S60-as platform 5. generációs készüléke például a következő tulajdonságokkal bír az elődeihez képest:

- továbbfejlesztett kezelőfelület, érintőképernyő támogatása

- 360x640 pixel felbontás támogatása
- érintés visszajelzés stb.

A Sony Ericsson más utat nézett ki magának és bár megtartotta a Symbian-os alapokat saját platformmal jelenik meg (UIQ). Ez lényegesen fejlettebb az S60-nál, nagy felbontású, érintés érzékeny kijelzőt használ, és támogatja a kézírás felismerést is. A Nokia-nak is van egy hasonló platformja, S90- néven. Minden készülék más kijelző felbontást használ, és csak a saját platformjára írt programokat képes futtatni (egy Nokia 6600-on nem indul el egy Nokia 9500-ra írt alkalmazás, és viszont).

3.2. Windows Mobile

A Microsoft a mobil eszközök terén is hódítani szeretne - a PC-s világhoz hasonlóan - , ezért a mobiltelefonokhoz és PDA- hoz egy saját platformot dolgozott ki, melynek neve a Windows Mobile.

Nézzünk meg egy- két operációs rendszert a Microsoft-tól:

- Pocket PC 2002 Premium Edition
- Pocket PC 2002 Professional Edition
- Pocket PC 2002 Phone Edition
- Windows Mobile 5.0 Pocket PC Professional Edition
- Windows Mobile 5.0 Pocket PC Premium Edition
- Windows Mobile 6.0 Professional és Windows Mobile 6.1 Professional
- Windows Mobile 6.1 Classic stb.

És a platformot támogató cégek a teljesség igénye nélkül:

- ASUS
- Motorola
- Samsung
- Sony Ericsson
- Hewlett-Packard stb.

A Windows Mobile 6.0 Professional főbb jellemzői:

- 320x320 és 800x480 pixel felbontás
- Microsoft Office Mobile 2007(Word, Excel, PowerPoint, OneNote Mobile)

- Térhatású grafikai elemek Windows Vista stílusban
- Windows Media Player stb.

3.3. Palm OS

PDA-khoz alkalmazott platform. Jellemző tulajdonságai:

- folyamatkezelés
- grafikus felhasználói felület 160x160, 320x320, 480x320 pixeles monokróm vagy színes kijelző
- 16 bit/pixel színes kijelző támogatása(65535 szín)
- audió lejátszása és rögzítése
- Videó lejátszás
- TCP/IP protokoll támogatás
- IEEE 802.11bWLAN támogatás
- Bluetooth, IrDA, USB, SD/MMC támogatás
- kézírás felismerés stb.

Palm platformot támogató készülékgyártók:

- IBM
- Qualcomm stb.

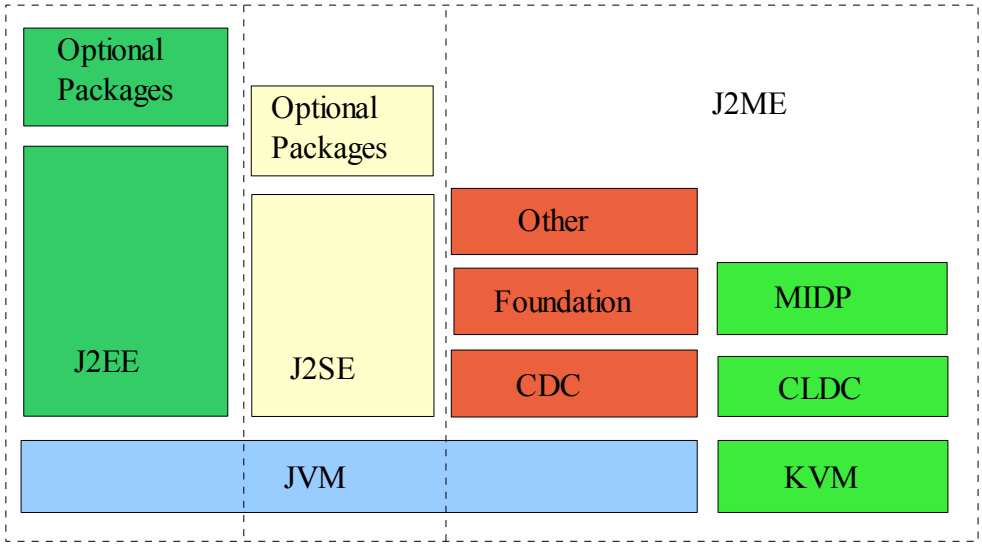
3.4. Android

Jelenleg még nem létezik olyan telefon amelyen Android platform létezne, mégis ígéretesnek tűnik az elmúlt egy év fejlődéseit tekintve. Az érdeklődőknek a Google biztosít fejlesztői környezetet. Ingyenes és nyílt forráskódú Linux alapú mobil platformról van szó, alapvetően Linux kernelre épül, csak speciális felhasználói programokat tud futtatni, ellentétben a hagyományos programokkal. A forráskódok Java nyelven készülnek, de a futtatáshoz nem a Sun virtuális gépét használja, hanem írtak hozzá egy saját futtató környezetet (Dalvik VM). Megfelelő hardverrel rendelkező mobil eszköz esetén olyan funkciók érhetőek el a platform által, mint az érintőképernyő, Bluetooth, EDGE, 3G, WIFI, GPS rendszer, iránytű stb.

4. A Java Technológia

A Java technológiát a Sun Microsystems fejlesztette ki. Maga a technológia egy programozási környezet és nyelv, amely az egyik legelterjedtebb, platformfüggetlen programozási nyelv.
A Java által kínált nyílt környezet lehetővé teszi, hogy a külső fejlesztésű alkalmazások folyamatosan letölthetőek legyenek a mobiltelefonra, újabb és újabb alkalmazások legyenek telepíthetőek a mobilkészülék teljes élettartama során. A Java kulcsfontosságú része a multimédiás kommunikációs alkalmazások fejlesztésének.
A Sun Microsystems 2006-ban nyílt forráskódúvá tette a Java technológiát. Ettől kezdve hozzáférhetőek a Java platform Standard Edition (Java SE), illetve a Java Micro Edition (Java ME) implementációi.
Amikor a köztudatban egyre többször halljuk azt, hogy ez meg ez „javas készülék”, akkor tulajdonképpen ez nem jelent más mint, hogy a készülék rendelkezik a Java technológiával.
A Java technológia a különböző termékek, a mobiltelefonoktól kezdve az intelligens kártyákon át a nagyvállalati alkalmazásokig és szuperszámítógépekig terjedő, széles palettájához kínál egységes szoftverfejlesztő platformot.
Nem mindegy, hogy egy Java programot egy mikro rendszerre, egy PC-re, vagy egy többprocesszoros szervergépre fejlesztettek ki. Itt természetesen nem csak a teljesítményre, hanem az egyes lehetőségek szűkülésére és bővülésére is gondolni kell. Éppen ezért a Java bevezette az úgynevezett Java platformokat. Ezek a platformok valamilyen általánosságban lefednek egy-egy felhasználási területet, maga a nyelv pedig az ottani igényeknek megfelelően szűkült, esetleg bővült.
Magát a technológiát három részre oszthatjuk:

1. **Standard Edition** (J2SE): A Java alap kiadása, személyi számítógépeken futtatható alkalmazások készítése és fejlesztése.
2. **Enterprise Edition** (J2EE): A Java vállalati kiadása, a hálózati és az ezzel kapcsolatos szerver/kliens oldali programozást segíti elő.
3. **Micro Edition** (J2ME): A Java mikro kiadása, olyan objektumokat tartalmaz amelyek a mobiltelefonokban képes élni és futni.



1. ábra Java Platform

4.1. J2SE

Ezt használjuk amikor PC-re írunk alkalmazásokat Kifejezetten munkaállomásokra (pl. PC) szánt változat. A Java itt indult az „Applet”- ekkel, majd az „önálló” asztali alkalmazásoknál is teret hódított. Ennek a platformnak kellően nagy memóriája és processzora van, illetve fontos a felhasználó számára kényelmes felhasználói felület is.

4.2. J2EE

Biztonságos szerveroldali Java alkalmazások fejlesztését a Java EE webszolgáltatásokat, komponensmodellt, menedzsment és kommunikációs API-kat kínál, ami a szolgáltatás orientált architektúrát alkalmazó nagyvállalati fejlesztésekben, illetve a következő generációs webalkalmazások terén ipari szabvánnyá minősíti. Az egész magját a következő komponensek alkotják:

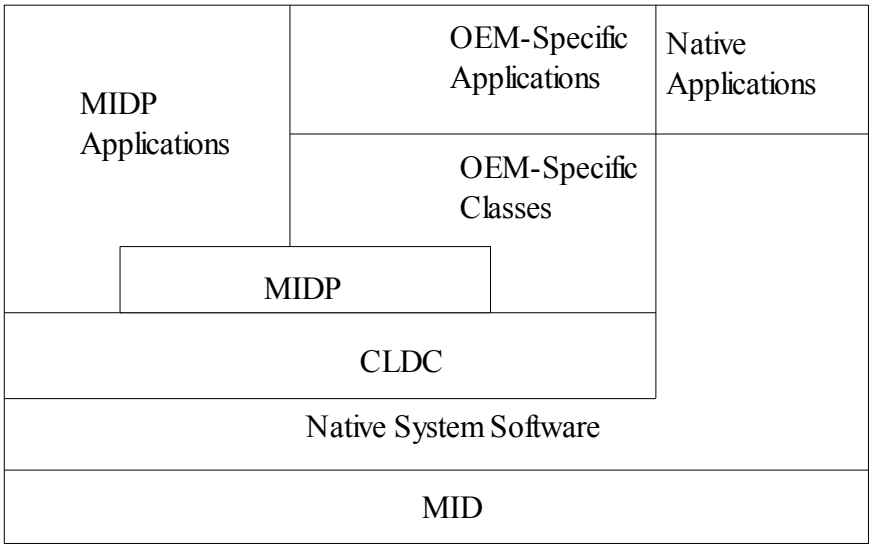
- Enterprise JavaBeans (EJBs)
- JavaServer Pages (JSPs)
- Java szervletek
- különféle interfészek a vállalat információs erőforrásaihoz történő csatlakozáshoz.

Üzleti alkalmazásra szánt változat, gyakran nagyon erős, többprocesszoros szervergépekhez, gigabájtos méretű memóriával. Leginkább webes alkalmazások 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.

4.3. J2ME

Kifejezetten a mobil környezetes fejlesztést teszi lehetővé. A Java Micro Edition-t elsősorban telepes üzemű, kis kijelzőjű, korlátozott beviteli lehetőségekkel és processzor teljesítménnyel rendelkező eszközökre fejlesztették ki (telefonok, PDA-k, személyhívók, stb.). A J2ME profilban az alkalmazások kezeléséért és telepítéséért a JAM (Java Application Manager) felelős. Az egyes alkalmazásokat `jar` fájlok képében telepíthetjük (ez lényegében egy ZIP fájl, amely tartalmazza az alkalmazásban szereplő osztályok bájtkódját, illetve információkat a JAM számára).

A J2ME platform biztosítja a Java technológia előnyeit a fenti eszközökön – rugalmas felhasználói interfész, robusztus biztonsági modell, hálózati protokollok széles skálája, valamint hálózati és offline alkalmazások támogatása.



2. ábra J2ME Platform architektúra

4.3.1. Konfigurációk

Mint tudjuk egy számítógép és egy telefon teljesítménye teljesen különböző. A JVM erőforrásigénye túl sok, hogy egy telefonon futtatható legyen, ezért a Java-ban két konfiguráció van definiálva. A konfigurációk a virtuális gépből és az osztálykönyvtárak egy részéből áll. A két konfiguráció:

CDC: Gyorsabb processzorral, nagyobb memóriával rendelkező eszközök konfigurációja. A nagyobb sávszélességet támogatja.

CLDC: Kevesebb memóriával rendelkező és kisebb teljesítményű eszközök konfigurációja, mint például a mobiltelefonok.

A két konfiguráció virtuális gépét KVM-nek (K Virtual Machine) nevezik. Futásához már 128 kilobájt memória és 16-bites 25 MHz-es processzor is elég.

A konfigurációkhoz szorosan kapcsolódik egy másik fogalom, a MIDP.

MIDP: A MIDP profil definiálja a kiegészítő API-k halmazát.

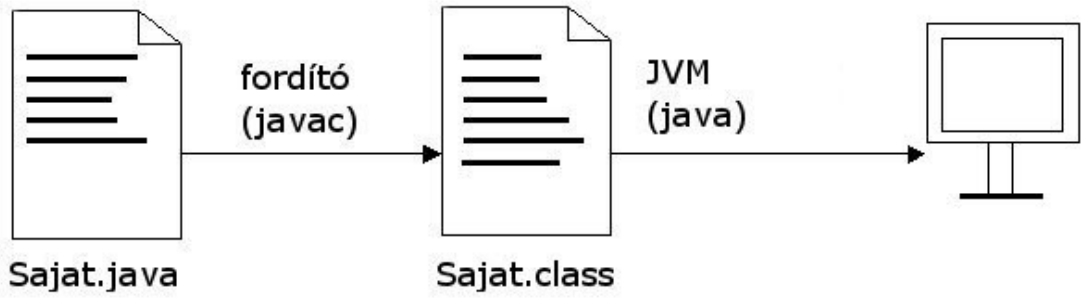
5. A Java nyelv

1991-ben a Sun Microsystems-nél egy kisebb fejlesztőcsoport dolgozott azon, hogy olyan „intelligens” háztartási eszközöket készítsen amely beépített mikrocsipekkel rendelkezik, ezáltal lehetővé téve az egymás közötti kommunikációt. Kezdetben a C++ nyelvel próbálkoztak, de hamar kiderült, hogy ez a nyelv nem alkalmas a feladatra. Ekkor a csapat egyik tagja név szerint James Gosling egy új nyelvet dolgozott ki.

Az új nyelv a C egy továbbfejlesztett verziója, de kihagytak belőle olyan elemeket amelyek a C nyelvben hibalehetőségek egész sorához vezetett. (Pl.: mutatók stb.) Az új nyelv az „Oak” (Tölgy) elnevezést kapta, mert állítólag James Gosling a dolgozószobája ablakából épp egy tölgyfára látott rá. Az új nyelvvel szemben támasztott legfőbb követelmény a platformfüggetlenség volt. Ez azt jelentette a Java nyelv szempontjából, hogy egy adott operációs rendszerre megírt program más platformon is futtatható legyen nem törődve azzal, hogy milyen hardver van a gépen. Ezt a JVM-el (Java Virtual Machine-Java Virtuális Gép) érték el. Ennek lényege egy, az operációs rendszer felett álló magasabb absztrakciós szintet jelentő szoftver.

A JVM tette lehetővé a különböző rendszerek közötti különbségek áthidalását. Minden jelentősebb operációs rendszerhez létezik ingyenesen elérhető JVM. Nem csak a személyi számítógépekre készült JVM, hanem mobiltelefonokra, PDA-kra is, amelyek képesek Java programot futtatni. A JVM-nek köszönhetően a fejlesztések is rugalmasabban történhetnek. Például egy Windows platformon megírt vagy fejlesztett alkalmazást gond nélkül futtathatunk más platformon, és viszont, de ezek feltétele, hogy telepítve legyen a platformokra a JVM (Java Virtual Machine).

Más programozási nyelvektől eltérően egy Java program fordítása és futtatása kicsit másképpen néz ki. Ezzel is mutatva a Java nyelv egyéniségét. Munkánkat a forrásállomány megírásával kezdjük. Példaként tekintsük az alábbi rajznak megfelelően ezt, `Sajat.java` formában. Egy program több `*.java` forrásállományt tartalmazhat. Ezek esetleg csomagokba (`Package`) vannak szervezve. Fordításkor a fordító(Compiler) egy bájtkódot állít elő (`Sajat.class`) és ezt a platformfüggetlen kódot értelmezi és futtatja a JVM. Ennek sematikus rajza a következőképpen néz ki:



3. ábra fordítás, futtatás

A fordítás egyszer történik meg, az értelmezés pedig minden egyes alkalomkor amikor futtatjuk a programot.

5.1. A Java nyelv szerkezete

Ebben a fejezetben csupán a Java nyelvre jellemző szerkezeteket és az objektumorientált programozással kapcsolatos ismereteket mutatom be. Majd ezekre mutatók példákat a programomból ezzel is szemléltetve a Java nyelv fontosságát és előnyeit más programnyelvekkel szemben.

A Java-ban a beépített, egyszerű adattípusú változók kivételével minden objektum. Az egyetlen összetett adattípus, amit használhatunk a tömb (*array*), ami teljes értékű osztályként használhatunk. Más programozási nyelvhez hasonlítva a Java nyelv nem tartalmaz globális változókat és globális eljárásokat. Minden adatot és eljárást valamilyen objektumhoz, esetleg osztályhoz tudunk kötni. A szabványos Java API-kból épül fel. Ez több ezer osztály részletes dokumentációja.
És most következzen a Java programozási nyelvre jellemző fogalmak tisztázása (amelytől más ez a nyelv),melyek véleményem szerint elhagyhatatlanok.

5.1.1. Objektumorientáltság

Az objektumorientáltság az egyik legdivatosabb programozási paradigma(alapelv). A Java programozási nyelv tisztán objektumorientált nyelv. Ez azt jelenti, hogy minden dolgot, tárgyat objektumként fogalmazunk meg és programozunk le. Az objektumorientált programozás alapja továbbá, hogy az adatokat és az őket kezelő függvényeket egy egységbe "zárjuk" (encapsulation – egységbezárás). Az objektumszemléletű megközelítés alapjaiban változtatja meg a programozással kapcsolatos gondolkodásmódunkat. Az elsődleges szerepet az adatok vagyis az objektumok töltik be. Az adatokon végrehajtott műveletek csak másodlagos szerepet töltenek be. A teljesség igénye nélkül általánosan ismertetem az objektumorientáltságot:

- Egy objektumorientált program együttműködő objektumok (`object`) összessége.
- A program alap építőkövei az objektumok Ezek a környezetüktől jól elkülöníthető, viszonylag független összetevők, amelyeknek saját viselkedésük, működésük és belső állapotuk van. Egy objektumra a környezetében lévő egyéb objektumok hatnak, amelynek hatására saját állapotuk megváltozhat.
- Minden objektum valamilyen osztályba (`class`) tartozik. Az osztályok megfelelnek az absztrakt adattípusnak, minden objektum valamely osztály példánya, egyede (instance). Az osztályok definiálják az egyes objektumok állapotát leíró adatszerkezetet, és a rajtuk elvégezhető műveleteket, az úgynevezett módszereket(`method`). Az egyes egyedek csak az állapotukat meghatározó adatszerkezet értékeiben különböznek egymástól, a módszereik közös.

- Az egyes osztályokat az öröklődés hierarchiába rendezi. Az öröklődés az a folyamat, amely segítségével egy osztály felhasználhatja a hierarchiában felette lévő osztályokban definiált állapotot (adatszerkezetet), és viselkedést (módszereket).

5.1.2. Osztályok

Az osztály felfogható egy típusként is, amely különböző változókat (adattagokat, más néven példány/egyedváltozókat) és módszereket (metódusokat/tagfüggvényeket) tartalmaz. Az előbbiek egy programban a tulajdonságokat, az utóbbiak pedig a viselkedéseket határozza meg. Az osztályokat elgondolhatjuk úgy is, hogy a hasonló tulajdonságokkal és viselkedéssel rendelkező objektumokat ugyanazon osztályokba soroljuk Az osztályokkal azt írjuk le, hogy milyenek lesznek azok az objektumok amelyeket az osztályból hozunk létre. Az osztály definíciója két részből áll:

- osztály deklaráció
- osztály törzs

Osztályokat a

```
class Name {
   Adattag1;
   Adattag2;
   ... stb.
   Tagfüggvény1(){
   ...
   }
   Tagfüggvény2(){
   ...
   }
   ... stb.
}
```

programszerkezettel definiálhatunk, ahol a `Name` az újonnan definiált osztály neve lesz, az objektumok belső állapotát és viselkedését pedig a kapcsos zárójelek közötti programrész írja le. Ezek lesznek az adattagok és tagfüggvények. Az osztálydeklaráció az osztály kódsorának

első sora, amelyben a `class` alapszó előtt, illetve az osztály neve után opcionálisan állhatnak még különböző módosítók. Ez az én esetemben a következőképpen néz ki:

```
public class Vaszon extends GameCanvas implements Runnable{
     A Vaszon osztály törzse a maga
     adattagjaival, metódusaival
     }
```

5.1.3. Objektumok

Az objektumok olyan programösszetevők, amelyek szoros egységbe foglalják az állapotukat leíró belső adatszerkezetüket és a rajtuk értelmezhető műveleteket. Ebben az értelemben az egyes objektumok nagyon hasonlítanak egy adott típusba tartozó értékekhez. Minden objektum önálló adattag készlettel rendelkezik. A Java-ban minden objektumnak meghatározott típusa van, azaz valamely osztályba kell tartoznia. Tehát az objektumok az osztályhoz köthetőek. Az objektumokra változókon keresztül hivatkozhatunk, pl. a :

```
Vaszon vaszon = new Vaszon();
```

Itt egy objektum definíciója látható.

5.1.4. Adattagok

Az osztály törzsében szerepelnek azok a változó deklarációk, melyek az egyes egyedek belső adatszerkezetét reprezentálják. Ezeket gyakran egyedváltozóknak nevezzük, jelezve, hogy az osztály minden egyede ezekből a változókból saját készlettel rendelkezik. Fontos megjegyezni, hogy az adattag(ok) deklarálása az osztály törzsében szerepel(nek), de a konstruktoron és a tagfüggvényeken kívül.

Például:

```
int utkozes = 0;
```

Itt egy `int` típusú `utkozes` nevű változót deklaráltunk, amely kezdőértékkel van ellátva.

5.1.5. Tagfüggvények

Objektumokon értelmezett műveleteket a tagfüggvények(metódusok, módszerek) testesítik meg, ezek definiálása:

```
void Name(formális paraméterlista){
az objektumokon végrehajtható utasítások leírása
}
```

A módszer nevét, visszatérési értékének számát, a paramétereinek a számát , típusát a módszer lenyomatának (signature) nevezzük. Vannak tagfüggvények melyeknek nincs visszatérési értéke, ilyen a fentebb definiált módszer. A tagfüggvények meghívásánál a formális paraméterek aktuális értéket kapnak. A Java-ban minden paraméterátadás érték szerint történik. Egy osztály belsejében azonos névvel, de különböző lenyomattal több módszert is definiálhatunk. Ilyenkor a fordító a paraméterek számából és típusából dönti el, hogy melyik módszert akarjuk meghívni.

Vannak azonban olyan tagfüggvények is amelyek visszatérési értékkel rendelkeznek, azaz valamilyen értéket adnak vissza pl.:

```
int Name() {
int x;
...
return x;
}
```

Ebben az esetben a visszatérési értéknek azonos típusúnak kell lennie a függvény típusával.

5.1.6. Konstruktorok

Egy osztályban definiált tagfüggvények közül kitűnnek az ún. konstruktorok (constructor), amelynek szerepe az objektumok létrehozásában, és belső állapotuk kezdeti értékének beállításában mutatkozik meg. A konstruktor az objektum életciklusa alatt pontosan egyszer fut le. Minden konstruktor neve megegyezik az osztálya nevével, visszatérési értéke pedig nincs. Kitüntet szerepe van a paraméter nélküli, ún. alap (default) konstruktornak, ugyanis ha mi nem definiáltunk magunknak, akkor a fordító automatikusan létrehoz egyet az osztályhoz.

Az így generált konstruktor a helyfoglaláson kívül nem csinál semmit.
A programomban saját magam hozok létre konstruktort ezzel felülírva az alapértelmezettet.

```
public class Vaszon extends GameCanvas implements Runnable{
...
     public Vaszon() {
     palya = new int [][]{
     }

     ...
     }
...
}
```

A konstruktorban létrehozásra kerül a pálya ami a játék hátterét adja, aztán betöltésre kerülnek a képek és így tovább.

5.1.7. Osztályváltozók és osztálymódszerek:

Amíg az egyedváltozókból minden példány saját készlettel rendelkezik, addig az osztályváltozókból osztályokként csak egy létezik. Mivel ezek nem egy példányhoz tartoznak, ezért nincs szükség konkrét objektumra, hanem az osztály nevével is hivatkozhatunk rájuk.

5.1.8. Csomagok

A Java platform csomagjai egy hierarchikus szerkezetet reprezentálnak. Csomagokban találhatóak a logikailag összetartozó osztályok definíciói. Az alapvető osztályok a `java.lang` csomagban, az I/O osztályok a `java.io`-ban találhatóak stb. Saját magunk is létrehozhatunk osztályokat annak érdekében, hogy osztályainkat egy egységbe csoportosítsuk és kezelhetőbbé tegyük.
Az összetartozó típusú osztályokat a csomag (`package`) segítségével egyetlen fordítási egységgé foghatjuk össze. Így osztály-könyvtárakat építhetünk fel, amelyben nagy szerep jut a láthatóság szabályozásának. A csomagot a forrásszövegben az első nem megjegyzés sorában

kell megneveznünk a `package` kulcsszó után.

```
Package Alkalmazas;
```

5.1.9. Osztályok importálása:

Ha más csomagban elhelyezett osztályt szeretnénk használni akkor annak több módja is lehet. Az egyik megoldás, hogy a felhasználás helyén a használatkor a csomagnévvel minősítjük az osztályt, vagy a forráskódunk elején az `import` kulcsszót használva megadjuk a használni kívánt osztályt. Nem csak osztályokat hanem csomagokat is importálhatunk. Ekkor a * karaktert használhatjuk annak érdekében, ha egy teljes csomagot szeretnénk beimportálni. Ez nem csak a MIDlet-re igaz, azaz nem csak a MIDlet-ünket tartalmazó `java` fájlra, hanem minden hozzá tartozó `*.java` kiterjesztésű fájlra.

```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
import java.util.Random;
```

5.1.10. Láthatóság

A programozók szabályozhatják az általuk elkészített osztályok, osztályok változóit, és módszereit, hogy milyen körben használhatók. Erre a célra ún. hozzáférést specifikáló módosítók használhatók. Ezek a módosítók a szó előtt szerepelhetnek és a következők lehetnek:

- nyilvános (`public`): bárhonnan elérhető, minden más osztály is láthatja, meghívhatja.
- privát (`private`): csak az osztályon belülről érhető el.
- baráti (`friendly`): ez az alapértelmezés, azaz ha nem írunk kulcsszót, akkor automatikusan ez állítódik be.
- védett (protected): az osztályból és annak leszármazottjából vagy csomagjából érhető el.

Osztályokra csak a `public` vagy `friendly` hozzáférés vonatkozhat, nyilvános osztályokat

a programunkban bárhol használhatunk. A `private` hozzáférésű változók és módszerek pedig csak az adott osztályon belül láthatóak.

```
public class Vaszon extends GameCanvas implements Runnable{
     private Image ufokep;
     private Sprite ufo;
     ...
     }
```

Például az én `ufo`-m és `ufokep`-em csak a `Vaszon` osztályban érhető el.

5.1.11. Öröklődés

Az objektumorientált programozás (OOP) egyik fontos sajátossága az öröklődés (inheritance). Az öröklődés mindig két vagy több osztály között fennálló kapcsolat. Az öröklődés során az egyik osztály örökli a másik osztály összes adattagját és metódusát. Az új osztályt származtatott (derived) osztálynak, míg a másikat ős osztálynak nevezzük. Ha egy új osztályt már valamelyik meglévő alapján akarunk definiálni, azt a következőképpen tehetjük meg:

```
class Osztaly extends SzuloOsztaly {
Osztaly törzse
}
```

Az így megadott, leszármazott osztály (`Osztaly`) örökli a szülője (`SzuloOsztaly`) tulajdonságait, azaz a belső állapotát leíró adatszerkezetet (egyedváltozóját), és a viselkedést megvalósító módszereit (tagfüggvényeit). Ugyanakkor a leszármazott osztályban módosíthatjuk is az örökölt módszert vagy módszereket és újakat is definiálhatunk. A programomban ez a következőképpen néz ki:

```
public class Vaszon extends GameCanvas implements Runnable{
     ...
     }
```

Az én általam létrehozott Vaszon osztály örökli a GameCanvas osztály összes adattagját és metódusait. Azaz már csak használnom kell az adattagjait és metódusait illetve a Vaszon osztályomban kiegészíthetem vagy felülírhatom őket.

Az öröklődéssel kapcsolatos fogalom még az általánosítás és a specializáló. A szülőből

örökölt adattagokat és tagfüggvényeket a `super()` referenciával érhetjük el. A Java az egyszeres öröklődést engedélyezi, azaz minden osztálynak csak egy szülője lehet.

5.1.12. Polimorfizmus

A polimorfizmus többalakúságot jelent, és amelyet többféleképpen is értelmezhetünk. Egyik ilyen értelmezése lehet, hogy különböző típusú objektumoknak lehet azonos lenyomatú módszere és e módszerek közül az objektum típusának megfelelő módszer kerül meghívásra. Ezt hívják metódus túlterhelésnek.

5.1.13. Kivételkezelés

Létezik egy utasításcsoport, amely a program működését nagyban befolyásolhatja, ez a `try -catch-throw` utasítás. A programok legnagyobb problémáját az egyes eljárások végrehajtása közben fellépő hibák, ún. kivételek (`exception`) kezelése jelenti. Egy `try` blokkba zárt utasításcsoporton belül bárhol előforduló kivétel hatására a program normális futása abbamarad és a vezérlés automatikusan a `catch` blokkra kerül. Itt a `catch` paramétereként megkapott hiba okát – ami egy `Exception` típusú objektum- kezelhetjük, ha akarjuk, vagy a lekezeletlen kivételeket a `throw` utasítással felfele passzolhatunk.

```
try {ufokep = Image.createImage("/ufo.png");
     csillagkep = Image.createImage("/csillag.png");
     csempek = Image.createImage("/hatter3.png");
    } catch(java.io.IOException e)
      {}
```

A programomban akkor keletkezik kivétel, ha a szükséges képeket - ami a program futásához elengedhetetlen - nem találja a program. A kivétel kezelését most nem kezeljük, aminek következménye a program automatikus leállása.

5.1.14. Interfészek

Az interfész nem más mint módszerek lenyomatának gyűjteménye, mégpedig olyan módszereké, amelyek egy osztályban való egyidejű meglétét a programozó fontosnak tartja. Az interfészben csak a tagfüggvények deklarációi szerepelnek, definíciói nem. Ezek abban az osztályban találhatóak, amelyek implementálják, megvalósítják az interfészt. Egy osztály több interfészt is megvalósíthat, ezáltal lehetővé téve a többszörös öröklődés érzetét adva. Nézzük meg hogy nézz ki a valóságban.

```
public class Vaszon extends GameCanvas implements Runnable{
    ...
    run(){
    ...
    }
    ...
    }
```

Implementálásra került a `Runnable` interfész, amelynek meg kell valósítania a `run()` metódust.

5.1.15. Megjegyzések

Más programozási nyelvekhez hasonlóan a Java-ban is alkalmazhatjuk a már jól megismert megjegyzéseket illetve kibővítjük még eggyel. A megjegyzéseket azért használjuk, hogy program működéséről információt szolgáltasson. A megjegyzéseket a fordító figyelmen kívül hagyja.

- //... : Egysoros információ elhelyezésére szolgál.
- /* … */ : Több soros információ elhelyezése.
- /** … */ : Dokumentációs megjegyzés, melyet a fordító ugyanúgy figyelmen kívül hagy, de a Javadoc segítségével automatikusan lehet generáltatni html dokumentációt.

6. Szoftvertervezés

Úgy gondolom, hogy egy szoftver tervezésekor a használni kívánt szoftverek mellett figyelembe kell venni a mobilkészülékek jellemzőit is, mert ez nagyban behatárolja a lehetőségeket. A PC-kkel ellentétben a mobiltelefonoknak:

- kisebb a kapacitása
- kisebb processzor sebességgel működnek
- kisebb a kijelzője

A termékpalettát tekintve igen gazdag a választék a piacon, amelyben olyan nagy gyártók versengenek egymással, minta a:

- Nokia
- Sony Ericsson
- Motorola
- Samsung
- Lg stb.

A cégek különböző mobilkészülékei között azonban találunk olyanokat, melyek mutatnak hasonlóságot, ilyen lehet például a kijelző mérete. Konkrét esetet nézve például egy Nokia készülék kijelzője 128 x 160 pixel méretű és egy Samsung kijelzője is lehet ugyanilyen méretű. De ugyanilyen hasonlóság fedezhető fel a kijelzők szín megvalósításaiban is. Ezek olyan tulajdonságok voltak melyek első szemrevételezéskor is feltűnnek egy felhasználónak. További hasonlóságok is vannak, mint például a kezelt fájlformátumok (3GP, MP3, MPEG-4, AAC stb.), de ezek a készülék áttanulmányozása után derülnek csak ki.

Amikor alkalmazást írunk akkor tudnunk kell pontosan a készülékünk specifikációját, melyet a készülékünk honlapján megnézhetünk. Saját készülékemet használva (Nokia 3110 Classic) az alkalmazás tesztelésére ez az én esetemben a következőképpen néz ki:

Nokia 3110 Classic Technikai specifikációja:

1. Operációs rendszer: Nokia OS
2. Fejlesztői platform: Series 40 3rd Edition, Feature Pack 2
3. Képernyő: Felbontás: 128 x 160
   Színmélység: 18 bit
4. Memória: Maximum felhasználható: 9 MB

Memória kártya: Micro SD

Max. JAR mérete: 1 MB

Halom(Heap) mérete: 2 MB

5. Java Technológia: MIDP 2.0

CLDC 1.1

JSR 135 Mobile Media API

JSR 172 Web Services API

JSR 177 Security and Trust Services API

JSR 184 Mobile 3D Graphics API

JSR 185 JTWI

JSR 205 Wireless Messaging API

JSR 226 Scalable 2D Vector Graphics API

JSR 75 FileConnection and PIM API

JSR 82 Bluetooth API

Nokia UI API

6. Video formátum: 3GPP formats (H.263)

H.264/AVC

MPEG-4

7. Audio formátum:

AAC, AAC+, eAAC+, MP3, MP4, WMA, AMR-NB, Mobile XMF, SP-MIDI, MIDI Tones (poly 64), True tones

A specifikáció nem teljes, csak néhányat emeltem ki. Minden készüléknek megvannak a maga specifikációi, amelyek áttanulmányozása után bátran nekifoghatnak alkalmazásokat írni. Ezek figyelembevétele nem elhanyagolható, ha azt szeretnénk és úgy látni a programunkat ahogy elterveztük.

6.1 Képernyő

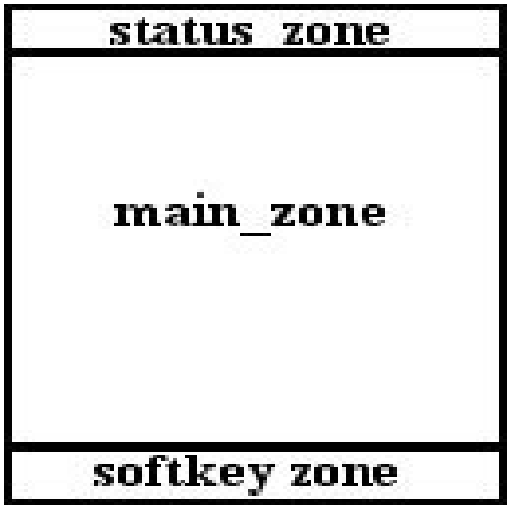
A kijelző méretét tekintve a következőek a legelterjedtebbek:

- 128 x 128 pixel
- 128 x 160 pixel
- 208 x 208 pixel

- 240 x 320 pixel stb.

A kijelző a következő általános komponensekből épül fel:

- **Status zone**: Az a terület ahol a telefon működésével kapcsolatos ikonok helyezkednek el. Pl.: akkumulátor töltöttségét visszajelző ikon, bluetooth bekapcsolását visszajelző ikon, ébresztőóra ikonja stb.
- **Main zone**: Itt jelennek meg az alkalmazásaink, melynek méretét jó ha tudjuk. Kivéve ha teljes képernyős az alkalmazásunk.
- **Softkey zone**: Billentyű parancsokhoz tartozó szöveg megjelenítése. Általában kettő vagy három billentyű gomb programvezérléséről dönthetünk.



4. ábra képernyő terület felosztása

Kezdetben a készülékek csak kétszínű (monokróm) kijelzővel rendelkeztek, majd a technológiai fejlődésnek köszönhetően megjelentek a színes kijelzőjű készülékek. Az idő előrehaladtával fokozatosan jelentek a még több színárnyalatot felvonultató készülékek:

- 4096 szín
- 65536 szín
- 262000 szín
- 16,7 millió szín

Az én mobiltelefonom 128 x 160 pixel méretű, a színkezelése18 bit-es (262144 színű), a softkey-ek száma 3 vagyis ennyi billentyűparancsot tud kezelni, a fejlesztői platform pedig Series 40 3. generációs csomag, amely ezeket a tulajdonságokat tudja kezelni. A Java

technológiák közül pedig kiemelném a MIDP 2.0-t és a CLDC 1.1-t. Ezek határozzák meg a készülék képességeit.

Egy alkalmazás megírását nagyban befolyásolja a feladat bonyolultsága, a programozó tudása és tapasztalata. Ezeket figyelembe véve nélkülözhetetlenek az olyan olyan szoftverek alkalmazása melyek nagy segítségére lehetnek a programozóknak. Elsősorban olyan szoftvereket mutatok be amelyek ingyenesek (tehát freeware esetleg regisztráció szükséges) és szabadon letölthetőek. Nézzük meg mivel is dolgozhatunk. Mi az amivel a munkánkat megkönnyíthetjük? Több lehetőségünk is van. Ezek közül csak néhány - általam ismert és a látókörömbe került- alternatíva kerül ismertetésre.

7. Fejlesztőkörnyezetek...

Kisebb programok írásakor és nagy projekt esetében is elgondolkodik az ember olyan fejlesztőkörnyezet használatán, amelyet kényelmesen használhat és a későbbiek során is szívesen visszatér ehhez. Ebben fejezetben ezt a célt tűztem ki.

Milyen előnyei vannak egy jó fejlesztőkörnyezetnek? Milyen elvárásaink lehetnek velük szemben? A fontosabb szempontok a következőek lehetnek:

Szintaxiskiemelés – különböző színnel jelöli meg a programunkban található elemeket, így könnyebb, áttekinthető kódot kapunk.

- Kódkiegészítés - nem kell a Java-ban egyébként megszokott hosszú változó- és metódusneveket beírni, az első néhány betű beírása után a fejlesztőkörnyezet kiegészíti azt.
- Integrált fordító- és futtatókörnyezet - nem kell parancssorral bíbelődni, pár kattintás és fordítja a kódokat. Ezen felül hibakereséssel és teljesítményelemzéssel is segíti a fejlesztésünket.
- Projekt-támogatás - összegyűjti és menedzseli az egy feladathoz tartozó fájlokat, kezeli az egymásra hivatkozásokat, kezeli a használt könyvtárakat, az ezekre való hivatkozásokat.
- Bővíthetők további modulokkal - ami pedig a végtelenségig bővíti a lehetőségeket.

Kezdő programozóknak és lelkes amatőröknek is csak ajánlani tudom az ilyen fejlesztő

környezetek használatát. Az induláshoz sok segítséget kaphatunk, ugyanis van bennük minta (Sample) amik tanulmányozása jó kiindulási alapnak bizonyulhat. Másrészt pedig a honlapjukon tutorialok egész sora található meg. (Angol nyelv ismerete ajánlott!)

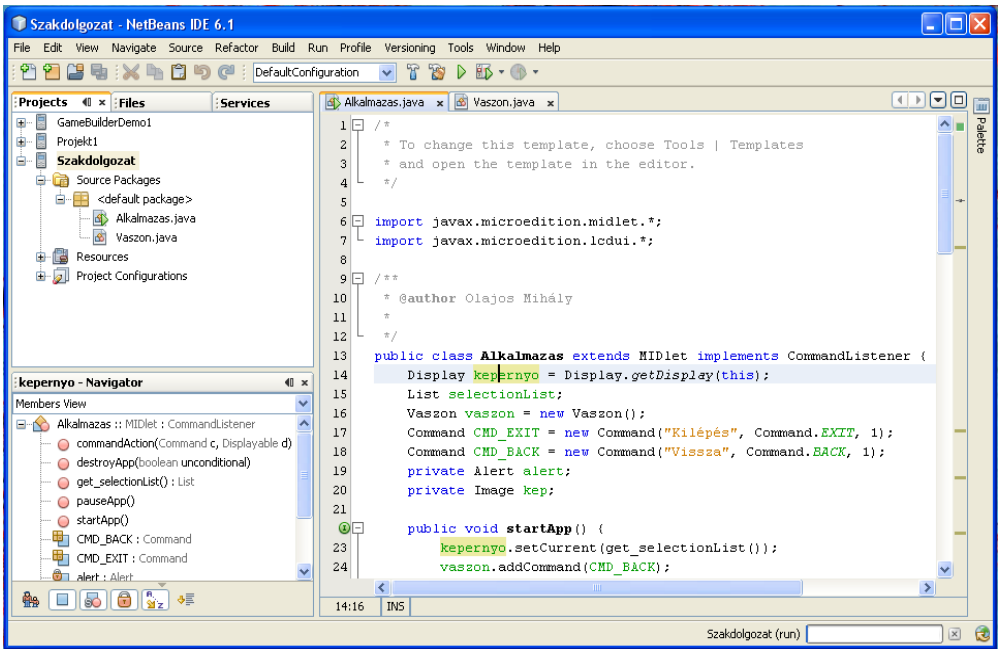
Fejlesztőkörnyezetek:

- NetBeans
- Eclipse

A fejlesztő környezeteket különböző platformra letölthetjük, mindenkinek ízlés dolga megítélni, hogy melyik platformon dolgozik. Lényeges különbség nincs, mert ugyebár ott van az a bizonyos platformfüggetlenség!

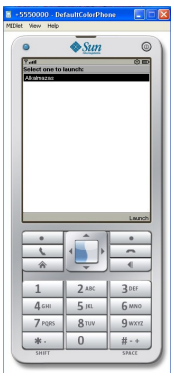
7.1. NetBeans

Az egyik ilyen fejlesztőkörnyezet amit legelőször megismertem és szívesen használok az a NetBeans. Egyszerűen azért, mert minden megtalálható benne amire egy programfejlesztés során szükségünk lehet. Ha netalán nincs benne, amire szükségünk lenne, akkor külön modulként letölthetjük és használhatjuk. A NetBeans IDE egy nyílt forrású, nagy teljesítményű, több platformon megjelenő Java integrált fejlesztőkörnyezetet kínál, amely felgyorsítja a szoftver alapú alkalmazások és webszolgáltatások fejlesztését. A programra rákereshetünk a Google-ban, vagy mehetünk célirányosan is a http://www.netbeans.info/downloads/index.php címre, ahonnan a Mobility oszlop alatti Download-al már el is kezdhetjük letölteni a mindig aktuális verziót. Jelen esetben a szakdolgozat írásakor a Netbeans IDE 6.1 a legaktuálisabb. Ez csupán a fejlesztőkörnyezet, melyhez szorosan kapcsolódik a JDK, vagyis a Java rendszer. Ennek meglétének ellenőrzése után, ha fent van a számítógépünkön akkor nagy megelégedettség tölthet el, egyébként pedig a http://java.sun.com/javase/downloads/index.jsp címről tölthetjük le. Itt keressük meg a JDK6 Update 7 csomagot, majd a mellette lévő Download gombbal elkezdhetjük letölteni



5. ábra NetBeans IDE

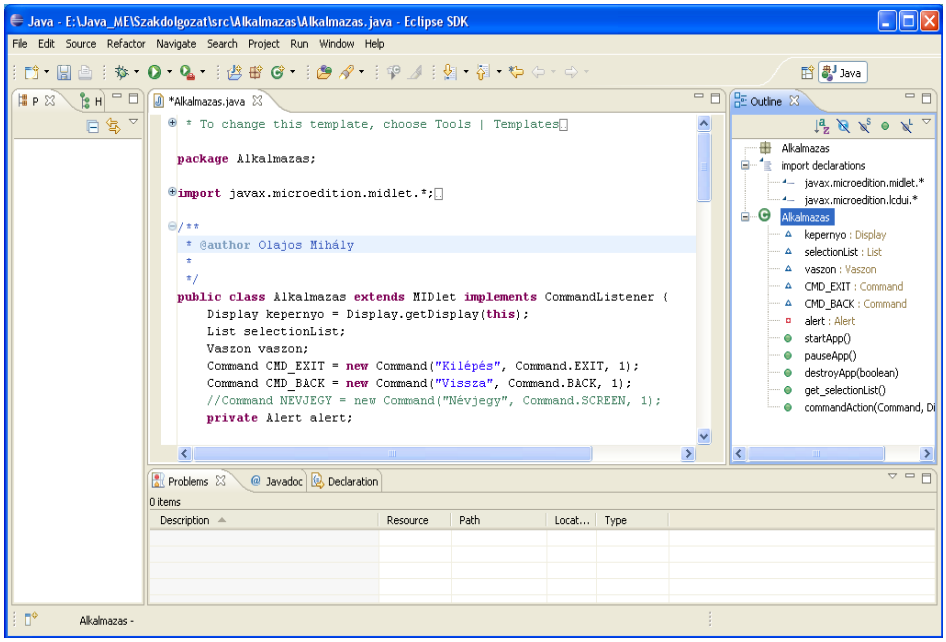
A programokat a beépített emulátoron próbálhatjuk ki vagy letölthetünk mi magunk is a készülékünk specifikációjának megfelelőt.



6. ábra: Emulátor

7.2. Eclipse

Az Eclipse egy nyílt forráskódú, platformfüggetlen keretrendszer, amelyet eredetileg az IBM fejlesztette ki. Ugyanúgy megvannak azok a tulajdonságai mint a NetBeans-nek. Az Eclips-el úgynevezett vastag kliens (rich client) alkalmazásokat lehet készíteni. A programot letölthetjük a következő címről: http://www.eclipse.org/downloads/



7.ábra Eclipse SDK

7.3. UML

Alkalmazásunk létrehozását megelőzheti egy olyan tervezés, amely nagy és bonyolult programoknál nélkülözhetetlen. Ezt a tervezést segíti vagy szolgálja a UML (*Unified Modelling Language*) azaz az Egységesített Modellező Nyelv. Az UML egy grafikus nyelv amellyel lehetőségünk van a probléma feltárására, megoldására, és dokumentálására.

A nyelv kialakulásában az alábbi szempontok játszottak szerepet:

- teljes (komplex) rendszereket akartak ábrázolni objektumorientált elv alapján
- explicit kapcsolatot kialakítani az elv és a végrehajtható kód között

- mind az ember, mind a gép számára érthető modellező nyelvet készíteni

Az UML jellemzői:

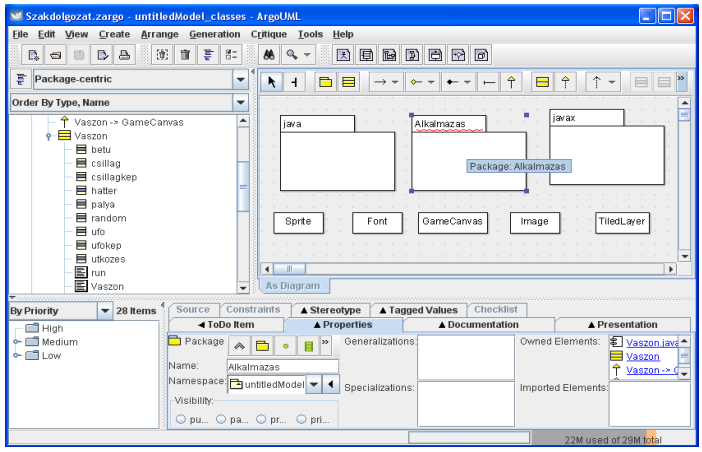
- Az UML-ben létrehozott modell grafikus szemléltetést nyújt.
- A nyelv lehetővé teszi modellek megalkotását különböző nézetekből.
- Alkalmas a probléma megoldásának pontos leírására.
- Alkalmas a probléma megoldásának dokumentálására, ezen belül:
  - a projektterv dokumentálására
  - a szoftverkészítés fázisainak dokumentálására
  - a prototípus dokumentálására

Nem csak szoftvermodellek leírására alkalmas, de munkafolyamatok, szervezetek, különböző üzleti tevékenységek stb. leírására is.

Az UML nyelv két jól elkülöníthető része a jelölésrendszer, és a metamodell. Ebből az első az, amivel általában találkozunk, mindazok a grafikus jelek, amelyeket a modellekben használunk. Ez az UML szintaxisa. A metamodell a nyelv szemantikája, osztály-diagrammokból áll, és a jelölésrendszert definiálja.

7.3.1. Ábra és diagramszerkesztő

Kisebb és nagyobb programok tervdokumentációit az UML nyelv segítségével is leírhatjuk. Ehhez nyújt támogatást a ArgoUML program. Használata nem bonyolult, kis gyakorlás után hamar belejön az ember. A program elérhetősége: http://argouml.tigris.org/



8. ábra ArgoUML

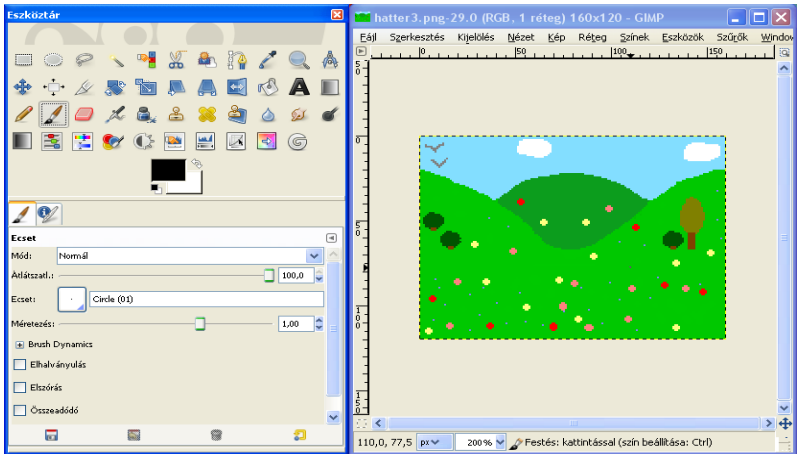
Az ArgoUML segítségével valósítottam meg az alkalmazásomhoz szükséges csomagokat, osztályokat, interfészeket és a köztük lévő kapcsolatokat. Az ArgoUML program területét három részre oszthatjuk. Az első területen (jobb felső) az ábrázolni kívánt elemeket (csomag, osztály, interfész, stb.) helyezhetjük el. A képen a programban használt csomagok láthatóak úgymint Alkalmazas, java, javax és a belőlük megvalósított osztályok Sprite, Font, GameCanvas, Image,TiledLayer.
A második terület (alul) az előbb említett munkaterületről kiválasztott elemek tulajdonságait, beállításait mutatja, hogy egy átláthatóbb képet kapjunk az elemről.
A harmadik területen pedig (balra fent) egy fa struktúrás megjelenést tesz lehetővé az ArgoUML. Az én estemben például az Alkalmazas csomag és annak tartalma látható vagyis a benne szereplő osztályok, adattagok, metódusok stb.

7.4. Grafikus program

Fontos szerepet játszanak a fejlesztés teljes időtartalma alatt hiszen az alkalmazások többségében használunk valamilyen képet, animációt stb.

7.4.1. GIMP

Ha a programunkban rajzot grafikát is szeretnénk alkalmazni akkor egy szintén ingyenes szoftverrel ezt is megoldhatjuk. Ezt letölthetjük a következő címről http:gimp-win.sourceforge.net/stable.html. Ennek mérete mindössze 15 Mb, tudását illetően pedig megközelíti a nagy fizetős programokét. A program használata nem nehéz, viszonylag gyorsan készíthetünk vele egyszerűbb képeket. Az alkalmazásomban használt Sprite-ok, TiledLayer elkészítését is ezzel a programmal végeztem



9.ábra Gimp 2.6

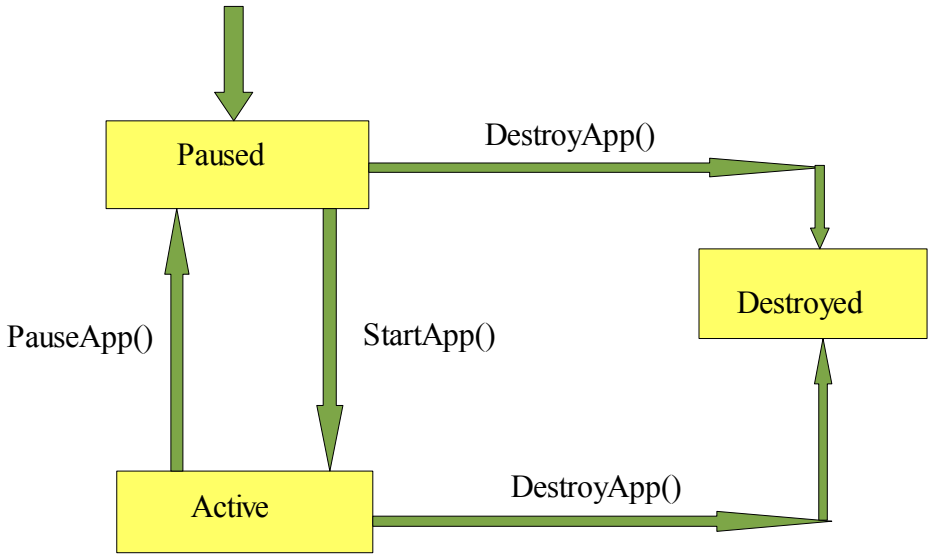
8. Egy Java program megvalósítása...

Ebben a fejezetben a saját alkalmazásom keresztül mutatom be a J2ME-t, a benne szereplő osztályokat és megvalósításukat. Az alkalmazásom tulajdonképpen egy játék alapjául is szolgálhat melyben egy ufót irányítva kell kikerülni a szembejövő csillagot. Ütközés esetén az ufó felborul, majd utána a program véletlenszerűen elhelyezi a csillagot és az ufót a képernyőre és folytatódik minden tovább. Az ütközések számát nyilvántartjuk és megjelenítjük a felhasználó számára. Az alkalmazás megalkotása nem okozhat gondot, ha ismerjük és megértjük a mögötte lévő osztályok viselkedését. És ez a lényeg! Hiába van nekünk több száz osztályunk adattagokkal és metódusokkal, ha nem tudjuk használni.

Az alkalmazásom egy MIDP alkalmazás, fő osztálya az `Alkalmazas` osztály, mely a `MIDlet` osztály leszármazottja. A `MIDlet` osztályt A `javax.microedition.midlet` csomag tartalmazza. Továbbá az `Alkalmazas` osztály implementálja a `CommandListener` interfészt. Ezzel kezeli a magas szintű eseményeket, mellyel meghatározza a parancsok viselkedését. A `Vaszon` (A Canvas osztály leszármazottja) osztály valósítja meg a képernyő megjelenítését. A `Vaszon` alacsony szintű eseménykezelő és alacsony szintű felhasználói felület.

A `MIDlet` osztály három absztrakt metódusát (`startApp()`, `pauseApp()`, `destroyApp()`) a származtatott osztályunknak implementálnia kell.

`StartApp()`: A függvény hatására a MIDlet aktív állapotba lép és lefoglalja a memóriaterületeket. Csak akkor lehet meghívni, ha szüneteltetett állapotban van.
`PauseApp()`: Ezen függvény meghívásakor a MIDlet szüneteltetett állapotba lép. Csak akkor lehet meghívni, ha a MIDlet aktív állapotban van. Erre akkor lehet szükségünk, ha az alkalmazás futása közben jön egy SMS, vagy esetleg egy hívás érkezik stb. Amint ilyen esemény érkezik akkor ezzel a metódussal kezelni tudjuk.
`DestroyApp()`: Ennek a függvénynek a hatására aktív vagy szüneteltetett állapotból az alkalmazás befejezését kérjük.



10. ábra Egy MIDlet életciklusa

A `MIDlet` csomag további elemeket is tartalmazhat, úgymint

- `notifyDestroyed()`
- `notifyPaused()`
- `resumeRequest()`

A `javax.microedition.lcdui` csomagban a `Display` osztály található. A `Display` reprezentálja a kijelző és a beviteli eszközök vezérlőjét. Definiáltunk egy `Display` típusú, és egy `Vaszon` típusú objektumot. Az `Alkalmazas` konstruktorában létrehozzuk a `Vaszon` típusú vaszon-t. Hozzárendeljük a `kepernyo`-höz a kijelzőt (ha még nem tettük meg) a `getDisplay(this)`-el, ezután a `setCurrent`-el megjelenítjük a `vaszon`-unkat, ami

ugye `Vaszon` típusú.

11. ábra Futás közben

12. ábra Ütközés

8.1. A Java ME világa és ami benne van...

Nos lássuk mi van benne! Osztályokat tartalmaz melyek egy hierarchikus rendszer alkotóelemei. Az osztályok további osztályokat is tartalmazhatnak. Ezen osztályok felhasználásával valósíthatjuk meg a programunkat, de természetesen ezeket felülírhatjuk, esetleg mi magunk is írhatunk saját osztályokat. Lássunk egy pár beépített osztályt: `Command` osztály, `Display` osztály, `Displayable` osztály, `Canvas` osztály, `Graphics` osztály, `GameCanvas` osztály stb.

8.1.1. GameCanvas

A `GameCanvas` osztály szolgáltatja az alapot a játék felülethez

```
public class Vaszon extends GameCanvas implements Runnable{
...
}
```

`void` `flushGraphics()`

Az off-screen buffert a képernyőre másolja.

void `flushGraphics(int x, int y, int width, int height)`

Az off-screen buffer megadott területét a képernyőre másolja.

```
FlushGraphics();
```

protected `Graphics getGraphics()`

Visszaadja a `Graphics` objektumot, amivel a `GameCanvas`-ra rajzolhatunk

```
Graphics g = getGraphics();
```

int `getKeyStates()`

Lekérdezi a fizikai játékgombok állapotát.

Használata:

```
while(true) {
     int lenyomottBillentyu = getKeyStates();
     if((lenyomottBillentyu & RIGHT_PRESSED) != 0) {
          if(ufoOszlop+25 < getWidth()){
               ufoOszlop += 5;
           }
     ... stb.
}
```

Tehát egy ÉS-el megvizsgálhatjuk, hogy az adott gomb le van-e nyomva.

void `paint(Graphics g)`

Megrajzolja a `GameCanvast`, nekünk kell implementálnunk

```
hatter.paint(g);
ufo.paint(g);
csillag.paint(g);
```

8.1.2. Layer

A Layer egy absztrakt osztály, ami a játék egy vizuális elemét reprezentálja, mint a `Sprite`, és a `TiledLayer`.

Közvetlen alosztályai: `Sprite`, `TiledLayer`

`int` `getHeight(); getWidth(); getX(); getY()`

A layer aktuális magassága, szélessége, x, y koordinátája.

`boolean` `isVisible()`

Visszaadja, hogy látható-e a layer.

`void` `move(int dx, int dy)`

A Layert a megadott dx, dy távolságra mozgatja.

`void` `setPosition(int x, int y)`

Beállítja a Layert úgy, hogy annak bal-felső sarka az (x,y) koordinátára esik

```
hatter.setPosition(hatterIndex, 0);
```

`void` `setVisible(boolean visible)`

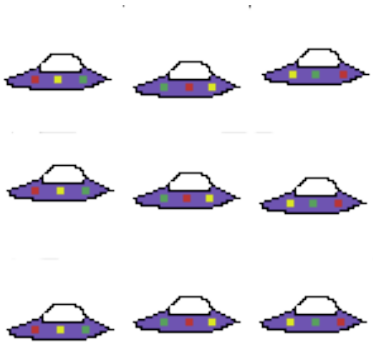
A Layer láthatóságát állítja.

8.1.3. Sprite

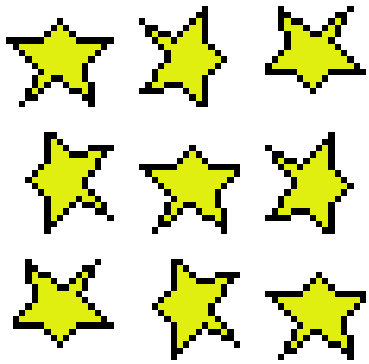
A `Sprite` egy vizuális alapja a játéknak, aminek megjeleníthető egy kockája a sok frame-ből, amit egy `Image`-ben tárolunk. A frame-ek változtatásával animálhatjuk a `Sprite`-ot.

Ha több mint egy frame-et használunk, akkor az `Image` egyenlő méretű frame-ekre lesz osztva. A kép elrendezése különböző is lehet. A frame-ek indexelve lesznek 0-tól kezdve.

```
Sprite ufo;
Sprite csillag;
```



13.ábra Ufó



14.ábra csillag

`Sprite(Image image, int frameWidth, int frameHeight)`

Létrehoz egy új `Sprite`-ot a megadott képből, a megadott szélességű és magasságú frame-ekre osztva a képet.

```
ufo = new Sprite(ufokep, 50, 50);
csillag = new Sprite(csillagkep, 20, 20);
```

boolean `collidesWith(Sprite s, boolean pixel)`

Megvizsgálja, hogy van-e ütközés a `Sprite` és a paraméterben megadott `Sprite` között.

Ha a `pixel` true, csak akkor van ütközés, ha nem átlátszó pixelek ütköznek.

Csak a Collision Rectangle-be eső pixeleket vizsgálja.

Ha a `pixel` false, akkor a Collision Rectangle-k fedését nézi.

```
if(ufo.collidesWith(csillag, false)){
...
}
```

void `defineCollisionRectangle(int x, int y, int width, int height)`

A `Sprite`-hoz egy behatároló téglalapot definiál, amit az ütközés-detektáláshoz használ

```
ufo.defineCollisionRectangle(1, 18, 42, 15);
csillag.defineCollisionRectangle(0, 0, 20, 20);
```

void `setImage(Image img, int frameWidth, int frameHeight)`

Megváltoztatja a képet, ami a `Sprite` frame-jeit tartalmazza.

void `defineReferencePixel(int x, int y)`

A `Sprite` referencia pixelét definiálja.

```
ufo.defineReferencePixel(25, 25);
csillag.defineReferencePixel(10, 10);
```

void `setRefPixelPosition(int x, int y)`

Beállítja a `Sprite` pozícióját úgy, hogy a referencia pixel az (x,y) koordinátákra esik.

```
ufo.setRefPixelPosition(ufoOszlop, ufoSor);
csillag.setRefPixelPosition(csillagOszlop,csillagSor);
```

void `setTransform(int transform)`

Beállítja a `Sprite` transzformációját.

```
ufo.setTransform(5);
```

void `setFrameSequence(int[] sequence)`

Beállítja a frame-ek sorrendjét a `Sprite`-hoz

```
ufo.setFrameSequence(new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8});
csillag.setFrameSequence(new int[]{0, 1, 2, 3, 4, 5, 6, 7,8});
```

void `setFrame(int sequenceIndex)`

Beállítja az aktuális frame-et a frame sequence-ből.

void `prevFrame()`

Az előző frame-re ugrik a frame sequence-ből.

void `nextFrame()`

A következő frame-re ugrik a frame sequence-ből

```
ufo.nextFrame();
csillag.nextFrame();
```

```
int getFrameSequenceLength()
```

Lekérdezi a frame sequence hosszát.

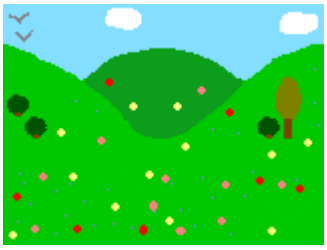
```
int getFrame()
```

Lekérdezi az aktuális frame index-ét a frame sequence-ből.

8.1.4. TiledLayer

A `TiledLayer` is egy vizuális elem, egy cellákból álló rács, ahol a cellákat feltölthetjük különböző képekkel.

```
TiledLayer hatter;
```



15.ábra Háttér

```
TiledLayer(int columns, int rows, Image image, int tileWidth,
           int tileHeight)
```

Egy új `TiledLayer`-t hoz létre a paraméterben megadott képből, amit a paraméterben megadott méretű képekre oszt.

```
hatter = new TiledLayer(palya[0].length, palya.length,
             csempek, 40, 40);
```

```
void setAnimatedTile(int animatedTileIndex, int
                     staticTileIndex)
```

Egy animált cellához rendel egy statikus értéket.

```
void setCell(int col, int row, int tileIndex)
```

Beállítja egy cella tartalmát a megadott indexre

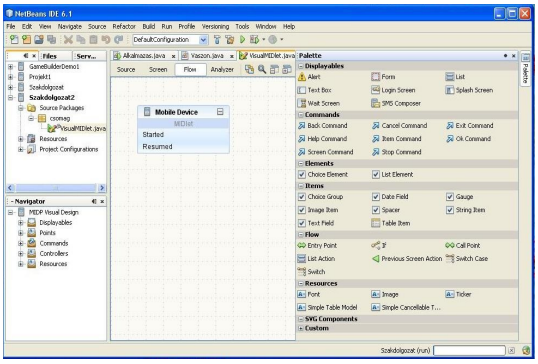
```
for(int i=0; i<palya.length; ++i)
    for(int j=0; j<palya[0].length; ++j)
        hatter.setCell(j, i, palya[i][j]);
```

```
void setStaticTileSet(Image image, int tileWidth, int tileHeight)
```

Megváltoztatja a képet, és paramétereit, amiből a cellákat töltjük fel.

8.2. Vizuális tervezés...

Programunkat elkészíthettük volna egyszerűbben is. Lehetőségünk van – a NetBeans-sel - vizuális tervezésre is, ahol csak össze kell rakosgatnunk a használni kívánt elemeket. A képernyő típusok közötti kapcsolatok ábrázolása és annak értelmezése leegyszerűsödik. A vizuális tervezés előnye az átláthatóság és kezelhetőség, a használt komponensek kódjának automatikus generálása stb. Ez pusztán vizuálisan mutatja meg a programunkat, a mögötte lévő eseménykezelésről magunknak kell gondoskodni. A gombok funkciójának működéséről, a képernyőn megjelenő form-ok viselkedéséről stb.



16.ábra Visual MIDlet

8.3. JAR és JAD

A JAR (Java Archive Resource) tartalmazza a MIDlet-et és a hozzá tartozó fájlokat. A JAR fájlban lévő manifest mondja meg minden MIDlet számára a MIDlet osztályt implementáló

osztály azonosítóját, nevét és ikonját. A MIDlet az az egyed, amelyet az alkalmazásvezérlő szoftver elindít, azaz létrehozza egy példányát. A JAR kiterjesztése `*.jar` és tartalma:

- Manifest leírás
- MIDlet(ek) osztály fájljai(t) és egyéb, a MIDlet(ek) által megosztott osztály(ok) osztályfájlokat.
- MIDlet(ek) erőforrás fájljai(t)

A Manifest nyújt információt a JAR fájl tartalmáról. Az én Manifest fájlom a következőképpen néz ki:

```
MIDlet-1: Alkalmazas, , Alkalmazas.Alkalmazas
MIDlet-Jar-Size: 233864
MIDlet-Jar-URL: Szakdolgozat.jar
MIDlet-Name: Szakdolgozat
MIDlet-Vendor: Vendor
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0
```

A JAD (Java Application Descriptor) egy alkalmazás leíró fájl amely lehetővé teszi az alkalmazás vezérlő számára, hogy az egész JAR fájl letöltése előtt eldöntse, hogy a MIDletek alkalmasak-e az eszközre. A fájl kiterjesztése `*.jad`

9. Összefoglalás és további lehetőségek

Úgy gondolom a szakdolgozatommal sikerült betekintést nyerni az alkalmazás fejlesztés kezdeti lépéseibe. Ismertetésre kerültek a fontosabb szoftver platformok, a fejlesztéshez szükséges programozási nyelv, a szoftvertervezés néhány szempontja.

A program továbbfejlesztése nyitott ami annyit tesz, hogy vannak olyan részek amivel a programot bővíteni lehet. Ilyen lehetőség például a grafikai tervezést illetően, hogy a programunkat egy `Splashscreen`-el látjuk el, aminek segítségével egy bevezető animáció indul. esetleg menüt tervezünk neki.

10. Alapfogalmak, szakkifejezések, rövidítések

- **JVM** – Java Virtual Machine: Java virtuális gép. A virtuális gép értelmezi (interpreter), és futtatja a bájtkódot. A fordítás egy alkalommal történik az értelmezés pedig minden alkalommal.
- **CDC** - Connected Device Configuration: Olyan konfiguráció, ami nagyobb mennyiségű memóriával rendelkező, nagy teljesítményű eszközökhöz készült. Általában TCP/IP protokollt használnak.
- **CLDC** – Connected Limited Device Configuration: kevesebb mennyiségű memóriával rendelkező és kisebb teljesítményű eszközökhöz készült konfiguráció. Ezen eszközök kommunikációja NEM a TCP/IP protokollra épül.
- **MIDP** – Mobile Information Device Profile: A CLDC alapú készülékekhez készült profil. Kis és középkategóriás mobil eszközökön futtatható programok utasítás készletét és egyéb jellemzőit határozza meg.
- **MIDlet**: A MIDP-ben meghatározott követelményeknek eleget tevő Java program.
- **JAR** – Java Archive Resource: Archivált tömörített Java program, *.jar kiterjesztésű fájl formájában találhatjuk meg. Ebben vannak a *.class fájlok és az egyéb erőforrások (pl.: képek, hangok stb.). Ezt töltjük le a mobiltelefonunkra..
- **JAD** – Java Application Descriptor: Egy *.jad kiterjesztésű fájl, ami információkat tartalmaz a MIDlet-ről, a funkciója pedig az, hogy a *.jar fájl letöltése előtt megállapítsa, hogy alkalmas-e az alkalmazás az adott eszközre.
- **JRE** – Java Runtime Enviroment: Java futtató környezet. Ez tartalmazza a JVM-t, ami futtatja a programokat. A futtató környezetnek két része van, a Java Plugin és a Java Web Start.
- **JDK** – Java Developement Kit: Java fejlesztő készlet. Ez szolgál arra, ha fejlesztgetni is szeretnénk, amely már tartalmazza a JRE-t is.
- **Java API**- Java Application Programming Interface: A Java platformon található több ezer osztály részletes dokumentációja.
- **Manifest** fájl: `*.mf` kiterjesztésű szöveges fájl a JAR-ban, ami a szvitről tartalmaz információt.
- **Suite**: MIDlet-ek együttese, ez van a JAR-ban.

11. Irodalom

- Juhász István: Programozás2
- Bátfai Norbert: Nehogy már... 2007
- Bátfai Norbert-Juhász István: Javát tanítok 2007
- Vartan Piroumian: Wireless J2ME Platform Programming 2002
- http://www.java.sun.com/
- http://www.java2s.com/
- http://www.forum.nokia.com/
- http://www.netbeans.org/
- http://www.gimp.hu/
- http://www.forum.nokia.com/devices/3110_classic
- http://www.codexonline.hu/CodeX4/alap/month/UML.htm
- http://hu.wikipedia.org/wiki/Symbian
- http://es.truveo.com/Kis-Gergely-Google-Android/id/2713209263
- http://www.ibm.com/developerworks/library/j-j2me/
- http://hu.wikipedia.org/wiki/Eclipse
- http://www.windowsceportal.hu/index.php?tartalom=fooldal