Debreceni Egyetem

Informatika Kar és Matematikai Intézet

Mobilprogramozás tanárként

Témavezető:
Bátfai Norbert
Számítástechnikai munkatárs

Készítette:
Török László
Informatika tanári szak

Debrecen
2007

Tartalomjegyzék:

Témaválasztás

Gyermekkoromban, ami meglehetősen régen volt, az „átlagos felhasználó” számára a számítástechnika, a számítógépek nagyon távoli dolgok voltak. Abban az időben a zenetanulás kötötte le figyelmemet. Felső tagozatos általános iskolás voltam, amikor először találkoztam egy Commodore Plus/4– es géppel. 13 – 14 évesen elsőre megfogott. Természetesen a játék volt az, ami lenyűgözött. Az első kábulat után egy számítástechnikai szakkör oszlopos tagjaként próbáltam kiismerni kis gépem számomra addig rejtett tudását. Ismerkedtünk az egyszerűbb programok elkészítésével. Nagy élmény volt, amikor először kiszámolta a gép helyettem, hogy „2 + 2 = 4”. Azután jöttek a komolyabb dolgok: adatok kezelése, grafika… na és persze a hangok…

Ekkorra a jövőm már nagyjából behatárolódott: felvettek egy zeneművészeti zeneiskolába. Ettől a ponttól kezdve szép lassan elsodródtam az informatikától, magával ragadt a zene.

Majd egy évtized múlva vettem egy számítógépet és újra jött az érdeklődés, ha lehet még nagyobb erővel…

Munkahelyemen, a zeneiskolában, érdeklődő gyerekekkel alakítottunk egy kis „zenei számítástechnikai szakkört”. Itt eleinte „csak” ismerkedtünk a számítógéppel és elsősorban felhasználói programok használatát gyakoroltuk. Olyan programokat, amelyek zenei ismereteket nyújtanak, amelyeket a gyerekek hasznosítani tudnak zenei tanulmányaik során is. Eljutottunk egyszerű programozási feladatok megoldásáig is.

Körülbelül ezzel egy időben itt a Debreceni Egyetemen választható tantárgyként objektumorientált programozást tanultam Bátfai Norbert tanár úr vezetésével. Közösen készítettünk egy programot mobiltelefonra. Ekkor merült fel bennem az ötlet, hogy a szakkörömön tanuló gyerekekkel készíteni kellene egy „mobilos” programot, amellyel a gyerekek nem csak a java nyelv és mobilprogramozás alapjait sajátíthatnák el játékos formában, hanem mindezzel még zeneelméleti tudáshoz is juthatnának.

2003 végén, 1,35 milliárd mobiltelefon előfizetés működött a világban. 2004 decemberében már 1,5 milliárdot regisztráltak. Egyes előrejelzések szerint 2010 – re ez a szám eléri a 2 milliárdot.

A gyerekeket lenyűgözi a mobiltelefon, amelyeken szinte kivétel nélkül futtathatók java alkalmazások. A fiatalok az átlagosnál magasabb fokú technológiai jártasságukból következően hamarabb veszik birtokba az új megoldásokat és szolgáltatásokat. Európában a fiatalok több mint 50 % - a rendelkezik mobiltelefonnal. Mindannyiuk számára nélkülözhetetlen kellékké vált a mobiltelefon, ami családjaikkal, barátaikkal való kapcsolattartásra és szórakozásra használnak. A gyerekek nagy része érzi úgy, hogy a mobiltelefon birtoklása biztonságérzetet nyújt számára.

Bevezetés:

Munkám során, a zeneiskolában tanártársaimmal együtt, szembekerültem azzal a ténnyel, hogy az iskolában tanuló gyerekek többsége nem érdeklődik a zeneelméleti oktatás iránt. Az elméleti tárgyat csak egy kötelező nyűgnek tekintik, melyen túl kell esni. Arra gondoltam, hogy egy mobilos zenei tárgyú alkalmazás használatával kellene a zeneelmélet tanulását serkenteni. Az alkalmazást pedig mi magunk fogjuk elkészíteni a számítástechnika szakkörön.

Ezen feladat megoldásához a szakkör tagjaival meg kell ismertetni a java programozási nyelvet. Meg kell ismerniük annak felépítését, működési sajátosságait. Meg kell ismerniük az objektumorientált programozást. Mindezt úgy, hogy a már korábban megismert programozási nyelvekkel össze is tudják hasonlítani.

A gyerekeknek meg kell ismerkedniük a java nyelvet alkotó fogalmakkal. Pl: objektum, osztály, csomag, példány, példányosítás, öröklődés, kiterjesztés, implementálás. Meg kell ismerniük a mobiltelefonon futtatható alkalmazások készítésének módját, valamint ezen programok lehetőségeit.

A programozási feladatokon túl, zeneelméleti tudás is gyarapítja ismereteiket. Az összeállított kérdéssorra nekik kell a helyes válaszokat megkeresni. Megtanulják, hogyan épülnek fel a különböző hangközök.

A kitűzött célok megvalósítását a java nyelv tanulmányozásával kezdjük. A következő lépés a mobilprogramozás. A program megtervezése, a kérdések összeállítása után következik a forráskód megírása.

Java és J2ME

Java

Mire is jó? Hát természetesen bármire - a java általános célú programozási nyelv -, a program nálunk fut, nem terheli az eredeti kiszolgálót, jópofa dolgokat rajzol, menükkel, párbeszédablakokkal tarkítja a képernyőnkre, hangokat ad, egyszóval életet lehel a letöltött dokumentumba. Persze ennél többet is tehet, például felveheti a kapcsolatot azzal a kiszolgálóval, ahonnan hozzánk került és a két gép tisztességes elosztott rendszerként buzgón kommunikálni kezd, ebből pedig bármi kisülhet.

A hagyományos böngészők képesek arra, hogy az általuk ismert, előre beprogramozott protokollokat használva felvegyék a kapcsolatot egy-egy kiszolgálóval és onnan adatokat töltsenek le, amelyeket, amennyiben a formátumát korábban ismerték, akkor megjelenítsék. Viszont ha az URL-ben (a kiszolgáló gépet és az ott tárolt információt megadó címben) ismeretlen protokollt adtunk meg, akkor a böngésző tanácstalan. Ha csak az adat formátuma ismeretlen, a helyzet egy fokkal jobb: a böngésző letölti a teljes adathalmazt, majd továbbadja a megjelenítés feladatát egy másik programnak, legrosszabb esetben tárolhatja azt a helyi lemezünkön, hátha majd egyszer megértjük, mi van benne. A Java-t értő böngészőkben viszont a kiszolgálóról letöltött java programok bővíthetik a felhasználható kommunikációs protokollokat és a megjeleníthető adattípusokat.

Az önállóan futtatható programok (application) és a böngészők által letölthető és futtatható "programkák" (applet) között nem nagy a különbség, legfeljebb csak a biztonsági meggondolások miatt a böngészők szigorúbban figyelik, korlátozzák a hálózatról letöltött programkákat. Önálló programnál ilyen védelem nincs, a felhasználó nyilván tudja, mit és miért indított el.

A hálózaton letölthető programok ötlete új színt hoz az elosztott rendszerekben divatos ügyfél-kiszolgáló paradigmába: a programok igény szerinti letöltése elmossa a kiszolgáló és az ügyfél közötti éles határokat.

Röviden az előzményekről

Az 1990-es évek elején valahol a SUN berkeiben elindult egy kevéssé fontos projekt azzal a céllal, hogy a cég betörjön a felhasználói elektronikai piac egy új, az ún. "okos", processzorral vezérelt, programozható (smart) készülékeket alkalmazó területére. E készülékcsalád jellegzetes képviselője a kábel-TV társaságok által használt vezérlő (set-top box). Az ilyen készülékek programozásához igény volt olyan architektúra-független technológiára, amely lehetővé tette a kész programok a hálózaton keresztül a készülékbe letöltését, megbízható futtatását.

A projekt kezdetben a C++ nyelvet használta, ám a fejlesztők ezt is, a többi, akkor hozzáférhető programozási nyelvet is alkalmatlannak találták a célkitűzéseik maradéktalan megvalósítására, hát új nyelvet terveztek maguknak. Kiindulási alapként a C++ nyelvet használták, kigyomlálva belőle - az általuk - bonyolultnak, nem megbízhatónak talált szerkezeteket, hozzáadva innen-onnan átvett, hasznosnak tűnő ötleteket, nyelvi elemeket.

A legenda szerint alkotója, James Gosling, nagyon szerette a fákat, különösen azt a tölgyet, ami az irodája előtt volt. Ennek tiszteletére szerette volna elnevezni Oak-nak, de ez a név már foglalt volt és nem akarta Oak2-nek keresztelni, mert az nem mutatott volna szépen. Tanakodott, tanakodott, mígnem kedvenc kávézójában, némi extra adag koffein kiváltotta élénk tudatállapotban megszületett az ötlet. Ránézett a kávéscsomagra, és a címkén ez állt: „Imported from Java”. Azóta Java néven ismert a találmány.

A Java megalkotásának célja, egy olyan C/C++ nyelvre hasonlító nyelv volt, amelyben könnyebb fejleszteni, mint a C/C++ nyelvekben. A nyelvnek teljes mértékben támogatnia kellett az objektum orientált gondolkodást (osztályok, öröklődés, polimorfizmus, stb.), valamint eleve felkészítették hálózati környezetben való futtatásra figyelem előtt tartva a biztonságot is (pl. internet böngészőben futó applet-ek és korlátozásaik). Fontos volt az a kitétel is, hogy a Java nyelvben készült programok lefordított változatai az úgynevezett Java virtuális gépen kell, hogy fussanak.

Hogy mi is az a virtuális gép? A klasszikus értelemben véve, 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 (pl. x86 vagy x64 utasításokkal). 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 bájtkódot képes értelmezni, vagyis az egyes Java programokat ilyen binárisba fordítja a fordító. A bájtkód egy átmenet a natív kód és a forrásfájl között. Kellően alacsony szintű (úgynevezett JVM Assembly), de nem annyira hogy a kódot ne lehessen hordozni.

A futtatás két módon lehetséges: a bájtkódot a virtuális gép a futtatás közben (JIT – Just In Time) értelmezi (vagyis minden JVM utasítást a végrehajtás pillanatában fordít az aktuális processzor által érthető változatra), vagy pedig a virtuális gép a végrehajtás előtt egy az egyben natív kódra fordítja a programot, majd ezt futtatja (Hotspot fordítás). Mindkettőnek ára van. Az értelmezéses megoldásnál a program a futtatás egésze alatt veszít teljesítményéből, míg előfordításnál az indulási idő nő meg. Szerencsére a mostani HotSpot fordítók többsége már kellően optimalizált ahhoz, hogy ez az idő ne tűnjön nagyon fel. A natív kód során is említettünk hátrányokat, itt is tegyük ezt meg: a bájtkódú programok vagy az értelmezés vagy pedig az előfordítás miatt sosem fognak olyan teljesítményűek lenni, mint a natív alkalmazások.

Mégis akkor 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. 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.

A Java jelenleg az 1.5 -ös változatnál jár, de az 1.6 -os változat kiadása is még idén esedékes. A Java osztálykönyvtára meglehetősen gazdag. Mindent megtalálhatunk benne kezdve a karakterlánc kezelésektől, a tömbkezelő függvényeken át az adatbázis kezelésig. Úgy is

mondhatnánk, hogy a Java meglehetősen produktív környezetet biztosít. Talán még érdemes megjegyezni, hogy a Java minden elemét jelenleg helyezik át open source (szabad forráskódú) licensz alá.

A nyelv legfontosabb tulajdonságai

Egyszerűség

A nyelv szintaxisa és szemantikája nagyban hasonlít a sokak által ismert C illetve C++ programozási nyelvekhez, megkönnyítve a kezdeti ismerkedést. A C++ nyelvből elhagytak néhány - programozói vagy fordítóprogram írói szempontból - bonyolult elemet. Kimaradt például az operátorok felüldefiniálásának lehetősége (operator overloading), a többszörös öröklődés. Eltűnt a goto utasítás, az automatikus típuskonverziók (coercion), az összetett adatszerkezetek közül a union, illetve C++-ban már amúgy is szükségtelen struct.

Azért szerencsére nem maradt ki minden a C++-ból, megmaradt például a futásidejű hibák lekezelésének mechanizmusa, az úgynevezett kivételkezelés (exception handling). Bár az „igazi programozók” megkönnyezik az eltávozott goto utasítást, de helyette címkézett ciklusokat, többszintű „break” és „continue” utasítást kaptunk.

Sokaknak elsőre furcsa, de a Java nem használ mutatókat (pointer), egy csapásra megszűntetve ezzel sok programozási hibalehetőséget. A programozó munkáját nagymértékben megkönnyíti az is, hogy a nyelv automatikusan felszabadítja a már nem használt tárterületeket (szemétgyűjtés, garbage collection).

Objektum-orientált

Manapság az objektumorientáltság divatos programozási paradigma (paradigmán értvén azon módszerek, elméletek, szabványok sokaságát, melyek együtt az ismeretek ábrázolásának egy lehetőségét adják - vagyis az objektumorientált programozás egy új, más szemszögből nézi a világot. A java lényegében a C++ objektumorientált tulajdonságait tartalmazza. A

programozó absztrakt adattípusként viselkedő osztályokat definiálhatnak, az osztályok műveleteket - módszereket - tartalmaznak, amelyek a rejtett adatreprezentáción (egyedváltozók) operálnak. Létrehozhatunk objektumokat, azaz egyes osztályokba tartozó egyedeket. Osztályok definiálásánál felhasználhatunk már meglévő osztályokat, az új osztály (a leszármazott) örökli a szülő adatait, módszereit. A módszerek hívásánál a meghívott módszer futási időben, az objektum aktuális típusának megfelelően kerül kiválasztásra (virtuális módszerek, polimorfizmus).

Az egyes osztályokban definiált változók és módszerek láthatóságát a C++-ban megismert módon – „private”, „protected” és „public” - lehet megadni.

Eltérést jelent a C++-hoz képest, hogy a java-ban a beépített, egyszerű adattípusú - numerikus, logikai és karakter típus - változók kivételével, minden objektum. Az egyetlen összetett adattípus, a tömb (array) teljes értékű osztályként viselkedik. A program nem tartalmaz globális változókat és globális eljárásokat, minden adat és eljárás valamilyen objektumhoz, esetleg osztályhoz kötődik.

A Java-ban minden módszerhívás - a fent említett statikus módszerek kivételével - virtuális. A C++-hoz hasonlóan lehetőségünk van az egyes objektumok típusát futási időben lekérdezni, sőt itt akár az osztályok forrásprogramban definiált nevét futás közben is felhasználhatjuk például objektumok létrehozására. Az osztályok mellett a java az Objective-C programozási nyelvből átvette az Interface fogalmat. Az Interface nem más, mint módszerek egy halmaza - adatszerkezeteket, egyedváltozókat nem tartalmaz -, amelyet egyes osztályok megvalósíthatnak (implementálhatnak). A java a C++-szal ellentétben nem engedi meg a többszörös öröklődést, viszont Interface-ek használatával, egyszerűbben, kevesebb implementációs problémával hasonló hatást lehet elérni.

Architektúra-független és hordozható

Napjaink hálózatait heterogén hardver- és szoftver architektúrájú számítógépek alkotják. A programok fejlesztését nagymértékben megkönnyítené, ha a forráskódból előállított program bármely architektúrán azonos módon futna. Ezen cél elérése végett a java nyelv nem

tartalmaz architektúra- vagy implementációfüggő elemeket. A C nyelvvel ellentétben a beépített adattípusok (pl. int) mérete - tárolásához szükséges memória mérete, a típus értelmezési tartománya - nyelvi szinten meghatározott.

Ahhoz, hogy a lefordított program, változtatás nélkül futtatható legyen különböző hardver architektúrákon, a fordítóprogram a programot nem egy konkrét processzor gépi kódjára, hanem egy képzeletbeli hardver - virtuális gép (virtual machine) - utasításrendszerére fordítja le. Az így létrejött közbülső, ún. Byte kódot töltjük le a célarchitektúrára, ahol a virtuális gépet megvalósító program értelmezi és hajtja végre.

A hordozhatóság nem csak a virtuális gépi utasítások, hanem a nyelv mellett szabványosított rendszerkönyvtárak szintjén is jelentkezik, ezek a könyvtárak valósítják meg a legfontosabb, operációs rendszerekhez kötődő feladatokat, mint például a be- és kiviteli rendszert, vagy a programok grafikus kezelői felületét.

Egy új architektúrán akkor futtathatók a java programok, ha már implementálták rá a virtuális gépet, beleértve a rendszerkönyvtárakat is. A virtuális gépet C-ben írták, a kód POSIX.1 szabványnak megfelelő operációs rendszert tételez fel, így viszonylag kis munkával hordozható. A hordozhatóság érdekes aspektusa a fejlesztői környezet hordozhatósága. A környezet egyes részei, mint például a fordítóprogram, nyomkövető eszközök, java nyelven íródtak. És ha a java programok hordozhatók, akkor az egész környezet is átkerült az új architektúrára.

Interpretált és dinamikus

Az implementált végrehajtás - kombinálva a klasszikus kapcsolatszerkesztő (linker) helyett futás idejű betöltéssel - a fejlesztési ciklust nagymértékben felgyorsítja. A java-ból eltűntek a C-ből ismert header állományok, feltételes fordítás, más programállományok fordítás közbeni beolvasása (#include). A lefordított programok tartalmaznak a szerkesztéshez szükséges minden információt. Elég csak a megváltozott állományokat lefordítanunk - nincs szükség a C-nél megszokott „make” programra, a forrásállományok közötti függőségek feltérképezésére

-, a program máris futtatható. Egyébként a java támogatja a nagybani programozást, összetartozó osztályok egy csomagba (package) foghatók, egyszerre fordíthatók. A láthatóság is figyelembe veszi a csomagokat. A kapcsolatszerkesztő helyét az úgynevezett osztály-betöltő (class-loader) veszi át, amely futás közben - ha szükség van rá - betölti az egyes osztályokat megvalósító lefordított, Byte kódú programállományokat. Az osztály-betöltő nem csak helyi állományokból, de szükség esetén a hálózaton keresztül is képes kódot letölteni.

Többek között a betöltő feladatának megkönnyítése végett a Byte kód tartalmaz a forráskódból átvett szimbolikus- illetve típus információkat. Az objektumorientált programozási paradigma egyik nagy ígérete, hogy általános célú, könnyen felhasználható osztály- könyvtárakat, "szoftver-IC-ket" hozhatunk létre a segítségével. Sajnos a C++ nyelv ilyen tekintetben nem váltotta be teljesen a hozzá fűzött reményeket. Nagyon nehéz olyan osztályokat tervezni, amelyeket később nem kell majd úgy módosítani, hogy ne kelljen azt például új - bár a programozók elől rejtett - adatkomponensekkel vagy módszerekkel bővíteni. Bár az osztály külső interfésze nem feltétlen változott meg, ám a C++ az osztály reprezentációját, tárbeli elrendezését kihasználó fordítási mechanizmusa miatt ilyenkor az összes, a megváltozott osztályt - akár csak öröklésen keresztül - felhasználó programot újra kell fordítani. Ezt hívják "törékeny alaposztály" (fragile base class) problémának. A java ezt a problémát úgy kerüli meg, hogy az osztályok adatkomponenseire, módszereire a Byte kódban is szimbolikusan hivatkoznak, a hivatkozások konkrét címekké kötése csak futási időben, a virtuális gépben történik meg.

A közbülső kódban megmaradt szimbolikus információk megkönnyítik a programok nyomkövetését. Sajnos az interpretált végrehajtásnak van egy nagy hátránya is, a programok sokkal - becslések szerint 10-20-szor - lassabban futnak, mint a gépi kódú megfelelőik. Bár a virtuális gép elég ügyesen lett kitalálva és a fordítóprogram is mindent megtesz azért, hogy ezt a virtuális architektúrát a lehető legjobban kihasználja, de még így sem vetekedhet a gépi utasítások sebességével. Ehhez még hozzáadódik a szimbolikus hivatkozások feloldásának ideje, a különböző betöltési- és futásidejű ellenőrzések, a tárgazdálkodási modellel járó szemétgyűjtési algoritmus futásához szükséges idő.

Robusztus és biztonságos

Ez a két fogalom a java esetében kéz a kézben jár. Robusztus egy nyelv, ha megakadályozza vagy futás közben kiszűri a programozási hibákat, biztonságos, ha megakadályozza, hogy rosszindulatú programok kerüljenek a rendszerünkbe. Mindkét célkitűzés eléréséhez gyakran hasonló módszereket használhatunk.

A robusztusság nyelvi szinten a szigorú, statikus típusosságban jelenik meg. Minden adatnak fordításkor jól definiált típusa van, nincsenek automatikus konverziók, az explicit konverzió csak kompatibilis típusoknál sikerül, egyébként legkésőbb futtatáskor programhibát (exception) okoz. A mutatók eltűnésével rengeteg potenciális hibalehetőség eltűnt a nyelvből. A dinamikus szemétgyűjtés megkímél bennünket a hasznos memória elszivárgásától (memory leak). Az egyedüli összetett adatszerkezet, a tömb használatakor a túlcímzést futási időben ellenőrzik. Az osztály-betöltő arra is figyel, hogy a módszereket megfelelő típusú paraméterekkel hívjuk meg.

A biztonság (security) a robusztussággal kezdődik, a fordító csak korrektül viselkedő programokat ad ki magából. Ez elegendő lehet önálló alkalmazásoknál, de a „programkák" letöltésénél ennél többre van szükség. Kezdjük azzal, hogy ki garantálja, hogy a letöltött Byte kódot valóban egy megbízható java fordító hozta-e létre? A Byte kód minden utasítása információt tartalmaz az operandusok típusáról, az osztály-betöltő - függetlenül attól, hogy a helyi háttértárról vagy a hálózatról tölt be - ellenőrzi, hogy a program megfelel-e a nyelv szabályainak. Eldönti például, hogy minden operandus valóban a megfelelő típusú, nem használjuk ugyanazt az adatot más-más típusúként is. Ellenőrizhető az is, hogy a módszerek a vermüket konzisztensen használják, valamint hogy a kód nem fér-e hozzá számára nem engedélyezett - a nyelv definíciója szerint láthatatlan, rejtett - adatkomponensekhez, módszerekhez.

A betöltő egy hivatkozott osztályt először mindig a helyi háttértárból próbál betölteni, csak akkor fordul a hálózaton elérhető kiszolgálóhoz, ha az osztály nincs meg a helyi rendszeren. Így elkerülhető, hogy trójai falóként valamelyik rendszerkönyvtár helyett azonos nevű, távolról betöltött programot futtassunk.

Ha a betöltött programkák átjutottak a betöltő konzisztencia ellenőrzésén, akkor a virtuális gép felügyelete alatt kezdenek futni, ez ellenőrzi, hogy a programok csak engedélyezett tevékenységet hajtanak végre. Szigorúan szabályozott - vagy teljesen tiltott - például helyi állományokhoz hozzáférés, tiltott más helyi programok indítása, jelentősen korlátozott a hálózaton felvetető kapcsolatok címzettje.

Többszálú

A programok jelentős része párhuzamosan végrehajtható részletekre - vezérlési szálakra - bontható. Így jobban kihasználható a számítógép központi egysége, a programok a külső - például felhasználói - eseményekre gyorsabban reagálhatnak. Az egymással kommunikáló, viszonylag laza kapcsolatban álló szálakra bontott feladat könnyebben áttekinthető, megvalósítható és belőhető. A többszálú programozáshoz a java nyelvi szinten biztosítja az automatikus kölcsönös kizárást - szinkronizált (synchronized) osztály, módszer vagy változó - , valamint a szálak létrehozásához, szinkronizációjának megvalósításához a rendszerkönyvtár tartalmaz egy úgynevezett „Thread” osztályt.

A java virtuális gép a szálak futtatásához prioritáson alapuló preemptív - de nem feltétlenül időosztásos - ütemezőt tartalmaz. Ez magyarázza például azt, hogy - a jelentős felhasználói igények ellenére - nem született meg a java virtuális gép a Microsoft Windows 3.1-es - csak kooperatív ütemezést tartalmazó - rendszerére.

Eleinte nem terjedt el, sem a nyelv a programozók, sem a kész programok a felhasználók körében: egy Java program futtatásához legalább egy nagyságrenddel erősebb gép volt szükséges, mint egy azonos feladatot ellátó natív program használatához. Sok helyütt olvasni olyan véleményeket, hogy a Java programok akár hússzor is lassabbak a gépre fordított programokhoz képest, azonban ez soha sem volt igaz. Ezek a hátrányok pár év alatt elkoptak, s jöttek az erősebb és gyorsabb gépek, amelyeken már nem tűnt lassúnak a Java program, a Sun Microsystems sokat dolgozott a lényegesebb és gyakran használt csomagok gyorsításán. Jelenleg ott tart a JRE, hogy egy C++ programhoz képest nemhogy lassabb,

hanem sok esetben gyorsabb futást eredményez, ha elég időt hagyunk a HotSpot beindulásához.

Idővel annyi ilyen gyerekbetegség és zavaró működés gyűlt össze, hogy jelentősen át kellett dolgozni a nyelvet. A fejlesztők hosszadalmas munkával elérték, hogy a kliens oldali alkalmazásokon túl szerver oldali programok csomagjai is használhatóvá váltak, így a Java az adatbázisokból felépülő weboldalak egyik meghatározó szereplőjévé nőtte ki magát nagyvállalati környezetben. A szerver operációs rendszerek diverzitásából előnyt kovácsolhatott a Java: már érezhető piaci veszteséget jelentett, ha a programokat csak egy rendszerre írták meg (ekkor jelent meg a .NET, amely a Java vetélytársa). Nem véletlen, hogy a nyelvet jelentősen átdolgozták, egységessé és könnyen tanulhatóvá alakították; illetve a harmadik fél által közzétett csomagok egységesen dokumentálhatók lettek a Sun által készített dokumentáló rendszerrel.

A felújított nyelv a Java2 nevet kapta. A grafikus felhasználói felület jelentős átdolgozásnak esett áldozatul, illetve az AWT (Abstract Window Tookit) mellett használható lett a Swing csomag is. Az előző verziók esetén még csak az AWT felületet lehetett használni, amelynek voltak előnyei és hátrányai. Az AWT felület összes komponense az akkoriban használt grafikus operációs rendszerek „közös nevezője” volt, mivel az elemek kirajzolását nem a JRE végezte, hanem az adott ablakkezelő rendszereknek a megfelelő komponenseit rajzolta ki.

Ez pár éve előnyös volt, mert sokat javított a programok futási tulajdonságain: az ablakok és elemeinek kirajzolása nem a virtuális gép feladata volt, így a grafikus felület használata nem volt sokkal lassabb, mint az adott platformra megírt más programoké. Enne eredménye az volt, hogy a felhasználó a számára megszokott felületet látta a program futása folyamán tetszőleges operációs rendszert használva. Az AWT hátránya egyenesen következik az előnyeiből: a kinézete függ a futtató környezettől, illetve a nagyobb probléma az, hogy a „közös nevező” miatt a komponenseinek száma és funkcionalitása enyhén szólva gyér és kötött. Ennek kiküszöbölésére kezdték el a Swing felületet fejleszteni, amely már az 1.2.0 JDK/JRE verzió óta elérhető – igazából a Java2 része – és használható igazi teljességében.

J2ME:

Egy zeneszám teljesen másképp szólal meg egy laptop hangszóróin, mint mondjuk a házimozi szettünkön. Hasonlóan igaz ez a Java programok különböző platformokon való futtatására: nem mindegy hogy egy Java programot egy kis mikró rendszerre fejlesztettek, vagy egy többprocesszoros szervergépre. És itt nem csak a teljesítményre, hanem az egyes lehetőségek szűkülésére és bővülésére is gondolni kell. Hogy elkerüljék a teljes káoszt, a Sun bevezette az ún. Java platformokat. Ezek a platformok valamilyen általánosságban lefednek egy-egy felhasználási területet; maga a nyelv pedig az ottani igényeknek megfelelően szűkítve esetleg bővítve lett. Járjuk át kicsit ezeket a platformokat:

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):

A 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 az ugyanilyen célból készült Personal Java utódja (teljes

mértékben le is váltotta azt). A J2ME virtuális gépe a KVM (Kilobyte Virtual Machine). Nevéből adódóan kilobájtos méretű (40-80KB!), 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 (ez lényegében egy ZIP fájl, amely tartalmazza az alkalmazásban szereplő osztályok bájtkódját, illetve információkat a JAM számára).

Java ME (Micro Edition):

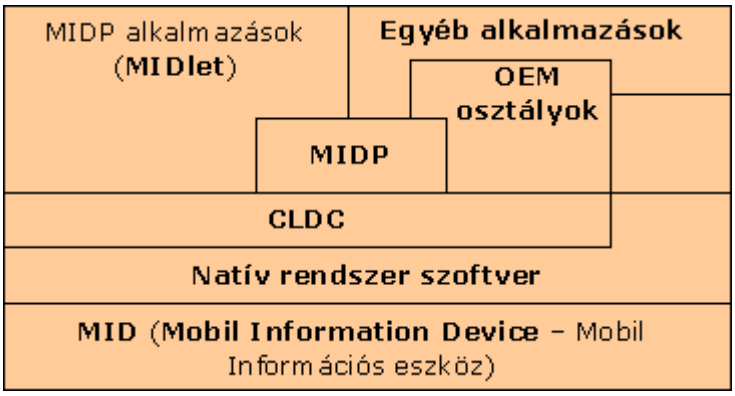
A J2ME környezet egy nagyon kicsi Java futtatási környezet, amely dinamikusan konfigurálható, hogy olyan környezetet szolgáltasson a fogyasztónak, amilyenre szüksége van egy alkalmazás futtatásához.

A konfigurációk a virtuális gépből, valamint az osztálykönyvtárak egy minimális halmazából állnak. Ezek szolgáltatják az alap funkciókat az eszközök egy sajátságos csoportjához, amelyekre jellemző a hálózati összekapcsolhatóság, és a csekély memória.

A két fő konfiguráció a Connected Limited Device Configuration (CLDC) és a Connected Device Configuration (CDC). A CLDC a kisebb a kettő közül, jóval kevesebb erőforrással rendelkező rendszerek (lassú processzor, korlátozott memória) számára a J2ME ezt technológiát használja. Ezek az eszközök tipikusan 16-32 bites CPU-val rendelkeznek és 128kB-512kB memória érhető el. A hálózati kommunikáció általában nem TCP/IP-re épül. A CLDC a CDC részhalmaza.

A CDC-t azokra az eszközökre tervezték, melyeknek gyorsabb a CPU-ja, nagyobb a memóriája és a hálózati sávszélessége. Ilyen eszköz például az Internet TV. Ez a technológia a klasszikus Java virtuális gépet használja, és egy jóval nagyobb alhalmazát képezi a J2SE platformnak, mint a CLDC. Ezek az eszközök általában TCP/IP-t használnak hálózati kommunikációra. Tulajdonképpen a CDC-t támogató készülékek 32 bites processzorral és minimum 2MB memóriával kell, hogy rendelkezzenek.

Magán a J2ME platformon belül is bevezettek különböző profilokat, hogy azok még inkább illeszkedjenek egy-egy hardver típusra.



A CLDC konfigurációra épülő profilok:

Ahhoz, hogy egy komplett környezetet szolgáltassunk a támogatott eszközök részére a konfigurációkat és magasabb szintű API-k halmazait (profilokat) kell ötvöznünk. Ez fogja definiálni az alkalmazások életciklus modelljét, a felhasználói interfészét, valamint a speciális eszköztulajdonságokhoz való hozzáférést.

A különböző elvárásoknak és követelményeknek megfelelően több profilt definiáltak. Azonban gazdaságilag is fontos, hogy az egy családba tartozó eszközök kompatibilisek legyenek, és tudjanak kommunikálni egymással (mint például a különböző gyártmányú mobiltelefonok).

Lehetséges az is, hogy egy készülék több profilt is támogasson, így aztán vagy készülék, vagy alkalmazás specifikusak lesznek. Egy piaci réteget megcélzó eszközöket gyártó, illetve alkalmazásokat fejlesztő cégek megegyeznek azokban jellemzőkben, amiket egy adott profil határoz meg:

A kialakult profilok a következők:

MIDP (Mobile Information Device Profile) profilt mobiltelefonokra, PDA-kra tervezték. Ez szolgáltatja a magját a mobiltelefon alkalmazásoknak. Magába foglalja a felhasználói megjelenítést, a hálózati összekapcsolhatóságot, tárolási módokat. A CLDC-vel kombinálva egy tökéletes JAVA futási környezetet alkot.

FP (Foundation Profile) a CDC-ben legalacsonyabb szintű profil. A hálózati kapcsolatot valósítja meg. Alkalmazható mélyen beágyazott megvalósításokban, felhasználói felület nélkül. A PP-vel és PBP-vel együtt felhasználható azoknál az alkalmazásoknál, ahol GUI-ra (Graphical User Interface) van szükség.

PP (Personal Profile) profil azokat a készülékeket célozza meg, amelyek a teljes GUI-t igénylik. Magába foglalja a teljes AWT-t (JAVA Abstarct Window Toolkit), felajánlja a WEB-es szolgáltatásokat, könnyen lehet a WEB-es alkalmazásokat futtatni.

PBP (Personal Basic Profile) profil a PP egy alhalmaza. Környezetet biztosít az olyan hálózatra kapcsolható eszközöknek, amelyeknek csak az alap grafikus szintet igénylik. Speciális alkalmazásoknak speciális grafikai eszközök (jármű telematikai rendszerek, TV SET-TOP BOX).

A többi kiadáshoz hasonlóan a J2ME is rendelkezik a JAVA előnyeivel:

Hordozható kód, ugyanaz a JAVA programozási nyelv, J2ME-vel írt alkalmazások működtethetők J2SE-vel és J2EE-vel.

J2ME a mobiltelefonok szemszögéből:

A J2ME mobiltelefonokra jutó profilját a MIDP képezi. Mint minden mást, a MIDP-t is folyamatosan fejlesztik. Jelenleg a MIDP2 az elterjedt, de már megkezdődött a MIDP3 fejlesztése is, így mobil fronton újabb követelményeknek megfelelő eszközök tűnhetnek fel hamarosan.

A MIDP változatok legfőbb jellemzői:

MIDP 1.0a (JSR 37):

A MIDP 1.0 az első MIDP változat, amelynek eredeti 1.0-ás változatát 2000. szeptember 1.-én adták ki, majd közel három hónappal a kiadás után elkészült a végleges 1.0a változat is. Mint látható, ez a szabvány az IT terület fejlődési tendenciájához viszonyítva elég régi, ám a későbbi profilok visszafelé kompatibilisek.

A MIDP specifikációkat az úgynevezett MIDPEG (Mobile Information Device Profile Expert Group) csoport készíti el, amelynek az 1.0-ás változatnál 22 tagja volt (Pl. Nokia, Ericsson, Siemens, Motorola, Samsung). A MIDP 1.0a a mai telefonok képességeihez viszonyítva meglehetősen alacsony követelményeket támaszt, ám akkoriban még csak nagyrészt monokróm kijelzős készülékek uralták a piacot, és ezek között is kevés volt Java képességű.

Képernyő méret: 96x54 pixel, 1 bites színmélység, megközelítőleg 1:1 arányú kijelzővel

Bemeneti eszköz: egykezes-, vagy kétkezes billentyűzet, vagy érintőképernyő

Memória: 128KB ROM MIDP komponenseknek, 8KB ROM a MIDP alkalmazások számára szükséges, hosszú távon tárolandó adatoknak (pl. beállítások). 32KB RAM a KVM halomterületnek (Heap).

Hálózati elérés: Kétirányú, nem feltétlen folyamatos, 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 elég kezdetleges I/O csomag, amely ekkor még csak http kapcsolatokat tudott létesíteni (javax.microedition.io csomag).

A MIDP 1.0-nak igen sok hiányosságot felrónak. Többek között, hogy nem támogatja a közvetlen grafikus műveletek elvégzését, így például a képen megjelenő képpontok színét sem lehet meghatározni. Nem programozhatunk Audio-t, nem kérdezhetjük le a billentyűk aktuális állapotát (csak listener-ek útján értesülhetünk róluk). Később ezeket egyes gyártók próbálták orvosolni kiegészítő csomagokkal. Ilyen például a Nokia UI, amellyel már előállíthatóak egyszerű hangeffektusok és a vibra motor is programozható (természetesen akkoriban még csak Nokia eszközökön).

MIDP 2.0 (JSR-118):

A MIDP 2.0 végleges változata 2002. november 5.-kén került kiadásra. A MIDP 2.0 hardver követelményei megegyeznek a MIDP 1.0-ával 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 titkosítást köteles ismerni. Kommunikációnál maradva az 1.0-ás változathoz képest az I/O csomag is kibővült olyan osztályokkal, amelyekkel például Socket vagy Stream kapcsolatokat is létesíthetünk a külvilággal.

A MIDP 2.0 felett is eljár lassan az idő, ezért fokozatosan bővítik újabb és újabb csomagokkal. Ilyen bővítés például az Infravörös adó, vagy a Bluetooth rádió használatának lehetősége is. Az egyes bővítéseket úgynevezett JSR-ek (Java Specification Request) formájában nyújtják be a résztvevő tagok a MIDPEG csoportnak. Ezek a bővítések többnyire opcionálisan kerülhetnek bele a MIDP 2.0-ás eszközökbe, hiszen nem tartoznak a MIDP 2.0 alapkövetelményeihez (talán majd a MIDP 3.0 már tartalmazni fogja őket).

A program szerkezete

Egy olyan mobiltelefonos alkalmazást készítünk, mely egy „elméleti” és egy „gyakorlati” részből áll. Az elméleti részben a felhasználónak kérdéseket teszünk fel, melyre legjobb tudásuk szerint válaszolhatnak. A kérdések megválaszolása után a program közli, hogy mennyire voltunk „okosak”. A gyakorlati részben két megszólaltatott hang közötti különbséget kell meghallani. A program itt a fülünket „minősíti”.

Import utasítás

Az osztálykönyvtárakból az `import` utasítással lehet egy vagy több osztályt, csomagot átvenni. Az `import` utasítás, a java és a javax osztálykönyvtárakból betölti ezeket az osztályokat, csomagokat, annak érdekében, hogy használhassuk őket. Midletünk forráskódja a következő `import` utasításokkal kezdődik:

```
import java.util.Random;

import javax.microedition.midlet.*;

import javax.microedition.lcdui.*;

import javax.microedition.media.Manager;

import javax.microedition.media.MediaException;
```

Ezekre a következők miatt van szükségünk:

A `java.util.Random` osztályra, a véletlenül előállított számok generálásához van szükségünk, melyet a hangközök készítésénél használunk majd fel.

A `javax.microedition.midlet` csomag tartalmazza a `MIDlet` osztályt, melyet a `ZeneElmelet` osztályunk kiterjeszt. A MIDlet osztály három absztrakt metódusát (`startApp()`, `pauseApp()` és `destroyApp()`) az örököseinek implementálni kell. A javax.microedition.lcdui csomagban található a `CommandListener` interfész, melyet a `ZeneElmelet` osztályunk implementál.

A `javax.microedition.lcdui` csomagban található a `CommandListener` interfész, melyet a `ZeneElmelet` osztályunk fog implementálni.

A `javax.microedition.media.Manager` osztályt fogjuk felhasználni a hangok megszólaltatására.

A `javax.microedition.media.MediaException` osztály a hanglejátszáskor adódó hibák miatt kell.

Felhasználjuk továbbá a következő osztályokat a MIDlet készítése során: `Alert`, `ChoiceGroup`, `Command`, `Dis\playable`, `Display`, `Form`.

Command

A Command osztály összefoglalja egy esemény szemantikai információit. Az eljárás, amelyet a Command aktivál, nincs benne az objektumban. Információkat csak a Command-ról kapunk, nem pedig arról, hogy mi történik akkor, ha aktiválódik.

Display

A Display reprezentálja a kijelző és a beviteli eszközök vezérlőjét.

ChoiceGroup

A ChoiceGroup választható elemek csoportja, melyek egy Form-on helyezhetünk el. A csoportot csoport elemei közül választhatunk egyet egyszerre, de beállítható úgy is, hogy engedélyezzen többszörös választást is. A megvalósításkor gondoskodnunk kell e módok kiválasztásáról.

Alert

Az Alert egy információs kijelző, ami tartalmazhat képet, szöveget és egy olyan eseménykezelőt, amely például egy másik Alert-et kezel.

Form

A Form egy olyan megjelenítő felület, amely tetszőleges tartalmazhat képet, olvasható szöveget, szerkeszthető szöveget, adatbeviteli mezőt, külön és akár egyszerre is.

A ZeneElmelet osztály a MIDlet osztályt terjeszti ki, és implementálja a CommandListener interfészt.

```
public class ZeneElmelet extends MIDlet implements CommandListener{
```

A MIDlet példánytagjai

```
    private Form form;

    private Display d;

    private int program=0;
```

Konstruktor

Itt beállítjuk az felhasználói interfészt, az első form-unkat, melyen szerepel midlet-ünk címe, valamint egy kételemű választási csoport, melyből csak egyet választhatunk. Itt inicializáljuk a Command típusú objektumokat. A Command osztály példányait a form-hoz adjuk, lényegében a menüpontok felkerülnek a telefon képernyőjére, és végül beállítjuk a parancsfigyelőt.

```
public ZeneElmelet(){
    String[] valaszt={"Kérdések","Hangközök"};
    d=Display.getDisplay(this);
    form=new Form("ZeneElmelet");
    form.append(new ChoiceGroup("Mit szeretnél? Kérdéseket
             vagy hangközöket? Válassz!",
             Choice.EXCLUSIVE,valaszt,null));
    form.addCommand(new Command("Kilep", Command.EXIT, 0));
    form.addCommand(new Command("OK",Command.OK, 0));
    form.setCommandListener(this);
}
```

Midletünk életciklusa

A `startApp ()` metódus megvalósításakor Midletünk képernyőjét az első `Form`-ra állítjuk.

A `pauseApp()` jelzi a `MIDlet`nek, hogy álljon meg, és lépjen Paused (várakozó) állapotba.

A `destroyApp()` pedig azt jelzi, hogy érjen véget, és lépjen Destroyed (befejezett) állapotba.

A `pauseApp()` és `destroyApp()` metódusokra jelenleg nincs szükségünk, ezért üres testtel felüldefiniáljuk őket.

```
public void startApp() {
      d.setCurrent(form);
   }
   public void pauseApp() {}
   public void destroyApp(boolean unconditional) {}
```

Metódusok:

HangkozFelismeres

Változók deklarálása és inicializálása

A `PlayTone` metódus működéséhez szükséges változók beállítása.

A hangok előállításakor az `ALSO_HATAR és a MAX_TAVOLSAG` szabja meg a hangok magasságát. A normál zenei „Á” hang a 69-es értéknek felel meg. Itt alapul a „C” hangot vettük, értéke 60. Ez az a hangmagasság amely „középfekvésben” található, tehát könnyedén, és hallhatóan megszólal egy átlagos kisméretű hangszórón is, mint amilyen a legtöbb mobiltelefonban található. A hangközök mivel a program gyerekek részére készült, a könnyebbség kedvéért egy oktávon belül szólalnak meg. A hangmagassági értékek eggyel való növelése, egy fél hang emelkedését jelenti. Egy oktáv 13 fél hangtávolságból épül fel. Innen a `MAX_TAVOLSAG`.

A HANGERO-t 100-as értékre vettem, ez a leghangosabb érték.

A HANGHOSSZ a lejátszott hangok hosszúságát jelenti milliszekundumban mérve.

A KEP változó, a program futása alatt segít azonosítani az éppen aktuális programállapotot. Azt, hogy éppen melyik hang lejátszásánál tartunk.

A hangkozok nevű sztring tartalmazza a válaszoknál kiválasztható hangközneveket.

```
    private static int ALSO_HATAR = 60;

    private static int MAX_TAVOLSAG = 14;

    private static int HANGERO = 100;

    private static int HANGHOSSZ = 1000;

    private Form formKezd,form1,form2,formValaszt,formZar;

    private ChoiceGroup valaszt;

    private int kep=0;

    private int h1,h2;

    private          String[]          hangkozok          =          {
"T1","K2","N2","K3","N3","T4","#4","T5","K6","N6","K7","N7","T8" };

    private Random random;
```

A HangkozFelismeres metódus a két hang előállításával kezdődik, oly módon, hogy addig állítunk elő véletlen egész számokat, amíg teljesül a következő feltétel:

Vesszük a véletlen szám abszolút értékét elosztjuk a MAX_TAVOLSAG-gal és veszzük a maradékot. Ez a szám 0-13-ig terjedhet. Addig ismételjük, amíg a második szám kisebb az elsőnél. Ha megvannak a számok, beállítjuk az formKezd-et HangkozFelismeres Néven, majd hozzáfűzzük egy üdvözlő szöveget. Inicializáljuk a Command típusú objektumokat. A Command osztály példányait a formKezd-hez adjuk, beállítjuk a parancsfigyelőt, és a formKezd felkerül a telefon képernyőjére.

```
public void HangkozFelismeres() {
    random=new Random();
    do{
        h1=Math.abs(random.nextInt() % MAX_TAVOLSAG);
        h2=Math.abs(random.nextInt() % MAX_TAVOLSAG);
    }
    while(!(h2>=h1));
    formKezd = new Form("HangkozFelismeres");
    formKezd.append(new StringItem(null, "Üdvözöllek a
                    hangközfelismerő programban!\n Az
                   első hang lejátszásához nyomj OK-t!"));
    formKezd.addCommand(new Command("Kilep", Command.EXIT, 0));
    formKezd.addCommand(new Command("OK",Command.OK, 0));
    formKezd.setCommandListener(this);
    d.setCurrent(formKezd); }
```

Kerdesek

Deklarációk és definíciók

```
private Command exitCommand;
Command Ertekel;
Command Help;
```

A kerdesek tömb tartalmazza a kvízjáték kérdéseit, a valaszok tömb pedig a megjelenő választási lehetőségeket. Továbbá meg kell még adnunk a megoldasok nevű tömbben, hogy a valaszok tömb elemei közül melyek a kérdésre adandó helyes válaszok. Ezt a kiértékeléskor használjuk fel.

```
    ChoiceGroup [] cg;

    int i,j;

     String kerdesek []  = {"Melyik operát írta Richard Wagner?","Ki
gyűjtött magyarországon először népdalt fonográffal?","Ki írta a
'Sors' szimfóniát?","Melyik Dúr hangsorban 4# az előjegyzés?","Mikor
halt meg Johan Sebastian Bach?","Melyik hang rezgésszáma 440?","Az
alábbi hangszerek közül, melyik nem rézfúvós hangszer?","Mi a
foglalkozása Leporellonak, a Don Giovanni című operában?","Hány
kürtversenyt írt W. A. Mozart?","Melyik országban játszódik az Aida
című opera?"};

    String           valaszok           [][]           =           {
{"Fidelio","Lohengrin","Tosca"},{"Kodály              Zoltán","Bartók
Béla","Vikár    Béla"},{"Haydn","Mozart","Beethoven"},{"E-dúr","Fisz-
dúr","A-
dúr"},{"1761","1750","1790"},{"C","E","A"},{"kürt","Szaxofon","tuba"
},{"Szolga","Földesúr","Bíró"},{"1","4","3"},{"Kína","Babilon","Egyi
ptom"}};

    int megoldasok [] = {1,2,2,0,1,2,1,0,1,2};
```

Inicializáljuk a `Command` típusú objektumokat.

Az új form azonosítója „`Kérdések`”. Egy for ciklusban végigmegyünk a `ChoiceGroup` típusú tömbön, melynek hossza megegyezik a kerdesek tömb hosszával. Ebben feltöltjük a tömböt, a `cg`-hez hozzáfűzzük a kérdéseket és a hozzájuk tartozó választási lehetőségeket a válaszok tömbből. A válaszok közül kérdésenként mindig csak egyet választhatunk. Majd a cg `ChoiceGroup` rákerül a `Form`-ra. A kerdesek és valaszok valamint a megoldasok tömb elemei szabadon változtathatók, tehát fel lehet tölteni őket új kérdésekkel és válaszokkal. Amire figyelni kell az, hogy a tömbök elemszámának meg kell egyeznie egymással, és természetesen a helyes megoldások kódjaira is figyelni kell. Különös tekintettel arra, hogy az elemek sorszáma 0-val kezdődik.

A `Command` osztály példányait a `form t`-hez adjuk, beállítjuk a parancsfigyelőt, és a `form t` felkerül a telefon képernyőjére.

```
public void Kerdesek() {
     exitCommand = new Command("Exit", Command.SCREEN, 1);
     Ertekel = new Command("Ertekel", Command.SCREEN, 1);
     Help = new Command("Help", Command.SCREEN, 1);

     Form t = new Form("Kérdések");
     cg = new ChoiceGroup[kerdesek.length];
     for ( i = 0; i < cg.length; ++i ) {
         t.append(new StringItem(null, kerdesek[i]));
         cg[i] = new ChoiceGroup("", Choice.EXCLUSIVE);
         for ( j = 0; j < valaszok[i].length; ++j ) {
             cg[i].append(valaszok[i][j], null);
         }
         t.append(cg[i]);
     }
     t.addCommand(exitCommand);
     t.addCommand(Ertekel);
     t.addCommand(Help);
     t.setCommandListener(this);
     d.setCurrent(t);
 }
```

commandAction

A commandAction indulásakor megvizsgálja, hogy a program változónak, amit a `MidLet` elején deklaráltunk, milyen az értéke. Ez az érték fogja azonosítani a `HengkozFelismeres` és `Kerdesek` metódusokat és dönti el a további lépéseket. A `MidLet` indulásakor a program = 0. Ha az első képernyőnél az OK gombot nyomjuk meg, akkor megvizsgálja, hogy az adott képernyőn a választási lehetőségek közül melyik van bejelölve. Kérdések vagy hangközök? Ezek a kiírás sorrendjében 0 vagy 1 értéket képviselnek. Választásunknak megfelelően, ha az érték 0, akkor a program változó átállítódik 2-re és meghívódik a `Kerdesek` metódus. Ha az érték 1, a program változó értéke 1 lesz, és a `HankozFelismeres` metódus lesz meghívva. Egyéb esetben (ha az OK helyett `EXIT`-et nyomtunk) a MidLet befejeződik.

```
public void commandAction(Command c, Displayable s){
    if(program==0){
        if(c.getCommandType() == Command.OK) {

     if(((ChoiceGroup)(form.get(0))).getSelectedIndex()==0){
                program=2;
                Kerdesek();
            } else {
                program=1;
                HangkozFelismeres();
            }
        } else notifyDestroyed();
```

Amennyiben a program értéke nem 0, hanem 1, az azt jelenti, hogy nem a `MidLet` elején vagyunk, hanem már a Hangkozfelismeres metódusban. Ilyenkor a telefon kijelzőjén már kint van a `FormKezd`, amelyen ki van írva az üdvözlő felírat, és az OK gomb megnyomásakor a `kep` változó értéke dönti el, hogy a hangközfelismerés részben éppen hol tartunk: 0 érték esetén az első hang fog megszólalni, amelyet a fentebb már említett, és itt meghívott `PlayTone` metódus fog megszólaltatni. A `kep` értékét növeljük egyel. Új ormunk a `Form1` HangkozFelismerés névvel, melyhez hozzáfűzzük a második hang megszólaltatására kérő üzenetet. Hozzáadjuk a `Form1`-hez a Kilep és az OK parancsokat. A parancsfigyelőt beállítjuk a `Form1`-re, és végül a `Form1`-et kiteszzük a kijelzőre.

```
        } else if(program==1){
            if(c.getCommandType() == Command.OK) {
                if(kep == 0) {
                    kep++;
                    try{

Manager.playTone(h1+ALSO_HATAR,HANGHOSSZ,HANGERO);
                    } catch(MediaException me) {}
                    form1 = new Form("HangkozFelismeres");
                    form1.append(new StringItem(null, "A második
                              hang lejátszásához nyomj OK-t!"));
                    form1.addCommand(new Command("Kilep",
                                          Command.EXIT, 0));
                    form1.addCommand(new
                              Command("OK",Command.OK,0));
                    form1.setCommandListener(this);
                    d.setCurrent(form1);
```

Ha a kep értéke 1, akkor az azt jelenti, hogy az első hang lejátszásán már túl vagyunk. Ekkor a PlayTone metódus a második hangot fogja megszólaltatni a fentebb már leírtak szerint. A kep értékét növeljük egyel. Új formunk a Form2. ehhez hozzáfűzzük a „Milyen hangközt hallottál kérdést, valamint a hangkozok tömb elemeit, melyek közül ki kell választanunk a szerintünk helyes megoldást. Természetesen csak egyet választhatunk. Form2-höz hozzáadjuk a kilep és Ok parancsokat, a Form2-re állítjuk a parancsfigyelőt, majd rátesszük a kijelzőre.

```
    } else if(kep == 1) {
     kep++;
     try{
Manager.playTone(h2+ALSO_HATAR,HANGHOSSZ,HANGERO);
                    } catch(MediaException me) {}
         form2 = new Form("HangkozFelismeres");
         form2.append(new ChoiceGroup("Milyen
                   hangközt hallottál?"
               ,Choice.EXCLUSIVE,hangkozok,null));
         form2.addCommand(new Command("Kilep",
                        Command.EXIT, 0));
         form2.addCommand(new
              Command("OK",Command.OK,0));
         form2.setCommandListener(this);
                 d.setCurrent(form2);      }
```

Ha a kep értéke 2, akkor már megtörtént mindkét hang lejátszása és már kiválasztottuk a szerintünk helyes választ is. A kep értékét növeljük egyel. Új formunk a `FormZar`. Ha az általunk választott megoldás megegyezik az első és második hang közötti különbséggel, akkor a `FormZar`-hoz hozzáfűzzük a „Válasz helyes!” üzenetet. Egyébként a „Rossz válasz!” üzenet és a helyes megoldás kerül ide. A `FormZar`-hoz hozzáadjuk a Kilep és Ujra parancsokat, beállítjuk a parancsfigyelőt, majd a `FormZar` rátesszük a kijelzőre.

```
else if(kep == 2) {

       kep++;

       formZar = new Form("HangkozFelismeres");

          if(((ChoiceGroup)(form2.get(0))).getSelectedIndex()==(h2-
h1)){

          formZar.append(new StringItem(null,"A

                                          válasz helyes!"));

        } else {

     formZar.append(new StringItem(null, "Rossz válasz!\n A helyes
válasz: "+hangkozok[h2-h1]));

                    }

     formZar.addCommand(new Command("Kilep", Command.EXIT, 0));

     formZar.addCommand(new Command("Ujra",Command.OK,0));

     formZar.setCommandListener(this);

     d.setCurrent(formZar);

     }
```

Ha a kep értéke 3, akkor `kep` értékét nullára állítjuk. A fentebb tárgyalt módon előállítjuk az új két hangunkat. Az új formunk a FormKezd, melyhez hozzáfűzzük a „Az első hang lejátszásához nyomj OK-t!” felszólítást, hozzáadjuk a Kilep és az Ok parancsokat, beállítjuk a parancsfigyelőt, és a `FormKezd`-et ráállítjuk a kijelezőre.

```
                else if(kep == 3) {
  kep=0;
do{
  h1=Math.abs(random.nextInt() % MAX_TAVOLSAG);
  h2=Math.abs(random.nextInt() % MAX_TAVOLSAG);
             }
while(!((h2>h1)&&(h1>=0)));
 formKezd = new Form("HangkozFelismeres");
 formKezd.append(new StringItem(null, "Az első hang
                      lejátszásához nyomj OK-t!"));
 formKezd.addCommand(new Command("Kilep",Command.EXIT,0));
 formKezd.addCommand(new Command("OK",Command.OK,0));
 formKezd.setCommandListener(this);
 d.setCurrent(formKezd);
          }
```

Amennyiben a program értéke 0, az új form a form lesz. Hozzáfűzzük a választható elemek listáját: kerdesek, valaszok, melyek közül egyet választhatunk. Hozzáadjuk a kilep és OK parancsokat. Beállítjuk a parancsfigyelőt, majd a form-ot rátesszük a kijelzőre.

Ekkor a már látott nyitókép tárul a szemünk elé.

```
            } else {
        program=0;
        String[] valaszt={"kerdesek","hangkozok"};
        d=Display.getDisplay(this);
        form=new Form("ZeneElmelet");
        form.append(new ChoiceGroup("Mit szeretnél? Kérdéseket vagy
hangközöket? Válassz!?",Choice.EXCLUSIVE,valaszt,null));
        form.addCommand(new Command("Kilep", Command.EXIT, 0));
        form.addCommand(new Command("OK",Command.OK, 0));
        d.setCurrent(form);
        form.setCommandListener(this);
```

Amennyiben a program értéke 2, akkor a kijelzőn a ZeneElmelet képét láthatjuk a választható elemeket tartalmazó felülettel. Ha Itt a kilépést választjuk, akkor MidLetünk alaphelyzetbe kerül: a program változó értéke 0 lesz. A form. hoz hozzáfűzzük a választható elemeket tartalmazó csoportot. Hozzáadjuk a kilep és az OK parancsokat, beállítjuk a parancsfigyelőt, majd rátesszük a kijelzőre.

```
} else if(program==2){
        if (c == exitCommand) {
        program=0;
        String[] valaszt={"Kérdések","Hangközök"};
        d=Display.getDisplay(this);
        form=new Form("ZeneElmelet");
        form.append(new ChoiceGroup("Kérdések vagy
              Hangközök?",Choice.EXCLUSIVE,valaszt,null));
        form.addCommand(new Command("Kilep", Command.EXIT, 0));
        form.addCommand(new Command("OK",Command.OK, 0));
        d.setCurrent(form);
        form.setCommandListener(this);          }
```

Ha a felhasználó a help-et választja, létrehozunk egy Alert típusú alert nevű változót, amelynek a példányosítás során a "Help" címet adjuk. Majd a setString () nevű beépített metódus segítségével kiírjuk a kijelzőre, hogy " A nyilakkal irányíthatsz, a select-tel válassz.\n Ha készen vagy összesítem az eredményt!".

```
    if (c == Help){
    Alert alert = new Alert("Help");
    alert.setString("A nyilakkal irányíthatsz, a select-tel
válassz.\n Ha készen vagy összesítem az eredményt!
    alert.setTimeout(Alert.FOREVER);
    d.setCurrent(alert);     }
```

Ha a felhasználó az Ertekel-t választja, létrehozunk egy Alert típusú alert1 nevű változót, amelynek a példányosítás során a "Kiértékelés" címet adjuk. A helyes válaszok számát 0-ra állítjuk. Egy for ciklus segítségével a megoldasok tömb elemeit összevetjük a felhasználó által választott válaszok indexeivel. Minden egyezés esetén növeljük eggyel a helyes válaszok számát, majd a setString () nevű beépített metódus segítségével kiírjuk a kijelzőre, hogy mennyi helyes válaszok száma. Majd Alert1-et ráállítjuk a kijelzőre.

```
if (c == Ertekel){
    Alert alert1 = new Alert("Kiértékelés");
    int helyesValaszokSzama = 0;
    for ( i = 0; i < cg.length; i++ ) {
    if ( megoldasok[i] == cg[i].getSelectedIndex() )
                        helyesValaszokSzama++;
                }
    alert1.setString("Helyes válaszok száma :
                    "+helyesValaszokSzama);
    alert1.setTimeout(Alert.FOREVER);
    d.setCurrent(alert1);
```

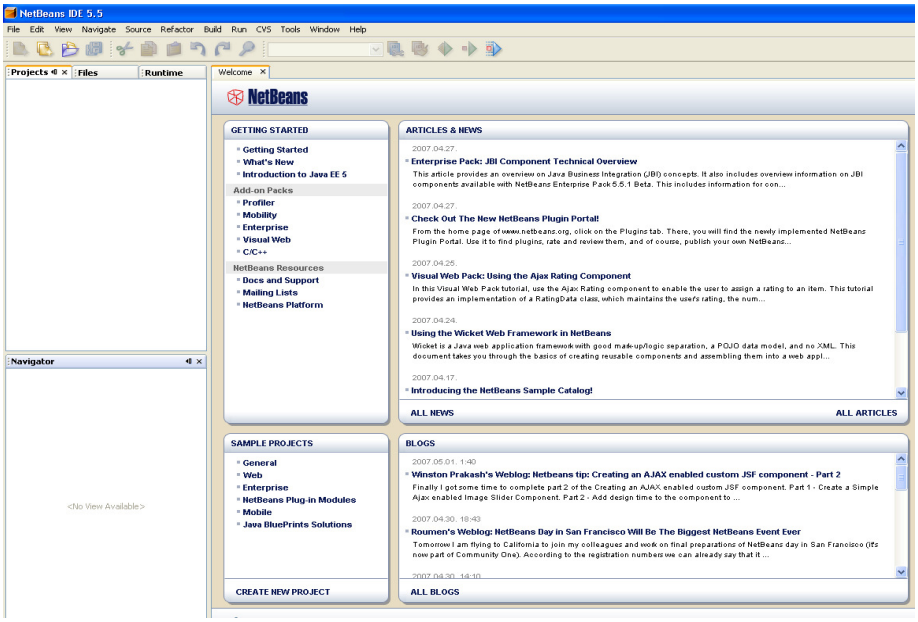
Fejlesztői környezet:

NetBeans:

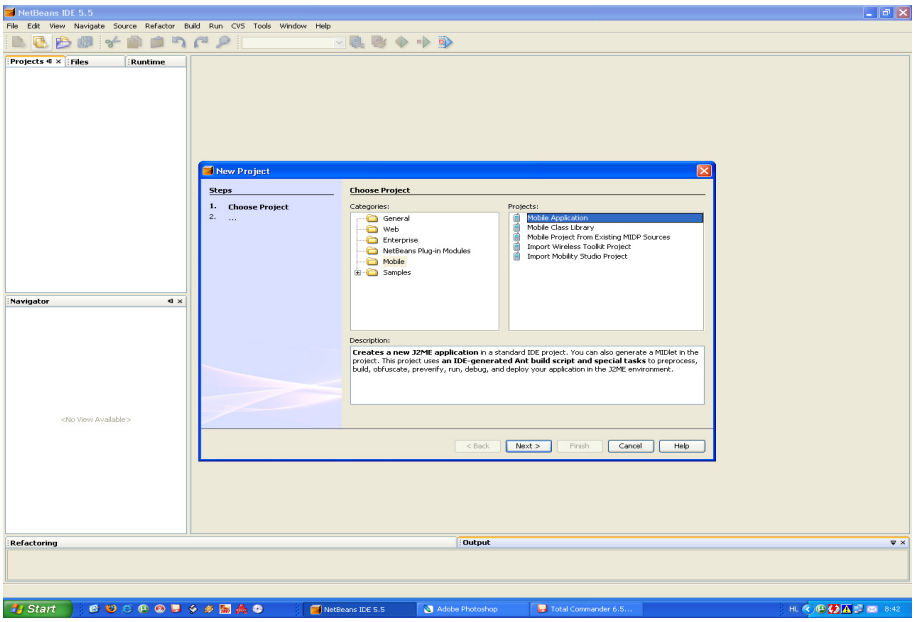
A NetBeans Developer 2.1 egy grafikus fejlesztőeszköz, amely a Java nyelven alapul. A program grafikus fejlesztőfelületet kínál a különböző alkalmazások, Appletek, vagy akár JavaBeanek elkészítéséhez, amelynek segítségével könnyebben, gyorsabban tudjuk fejleszteni saját programjainkat. Működése nagyon hasonló más grafikus eszközökhöz.

A http://java.sun.com/javase/downloads/netbeans.html címről, programot ingyenesen le tudjuk tölteni a. Ekkor a netbeans-szel együtt hozzájutunk a JavaSE legfrissebb, 1.6-os verziójához is. Ahhoz, hogy mobileszközre tudjunk fejleszteni, telepítenünk kell a jdk-1_5_0_11-windows-i586-p.exe modult.

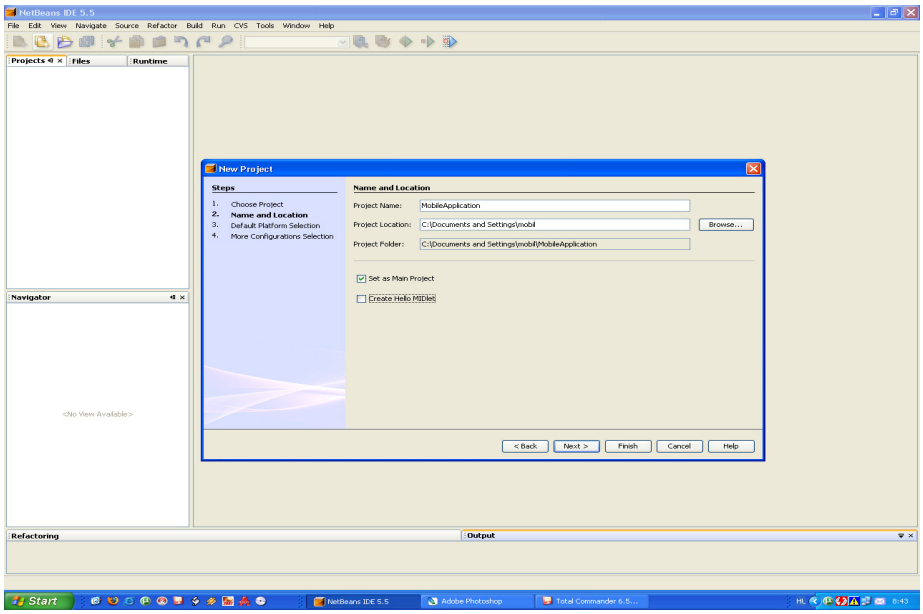
Telepítések után, az Netbeans indításakor egy üdvözlőképernyő tárul elénk, amit nyugodtan bezárhatunk.



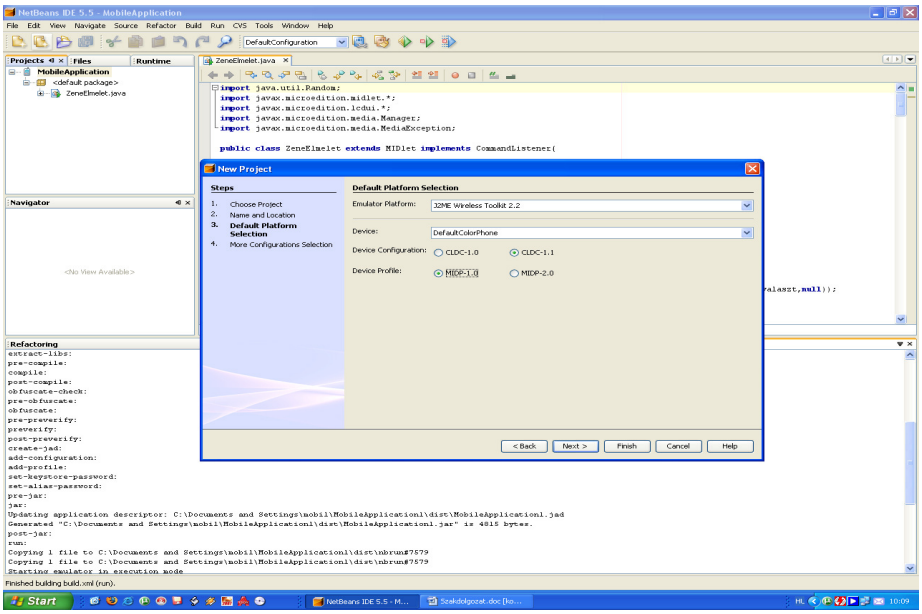
A beépített varázsló segítségével létre kell hoznunk egy új projektet:



Az Új Projekt varázsló párbeszédpaneljén be kell állítani, hogy agy mobil alkalmazást szeretnénk létrehozni. A tovább gomb megnyomása után, nevet kell adni projektünknek. Mi az alapértelmezettként felajánlott „ MobileApplication” nevet használjuk, és a „Create Hello MIDlet” jelölőnégyzetből kivesszük a pipát, mert arra nem lesz szükségünk.



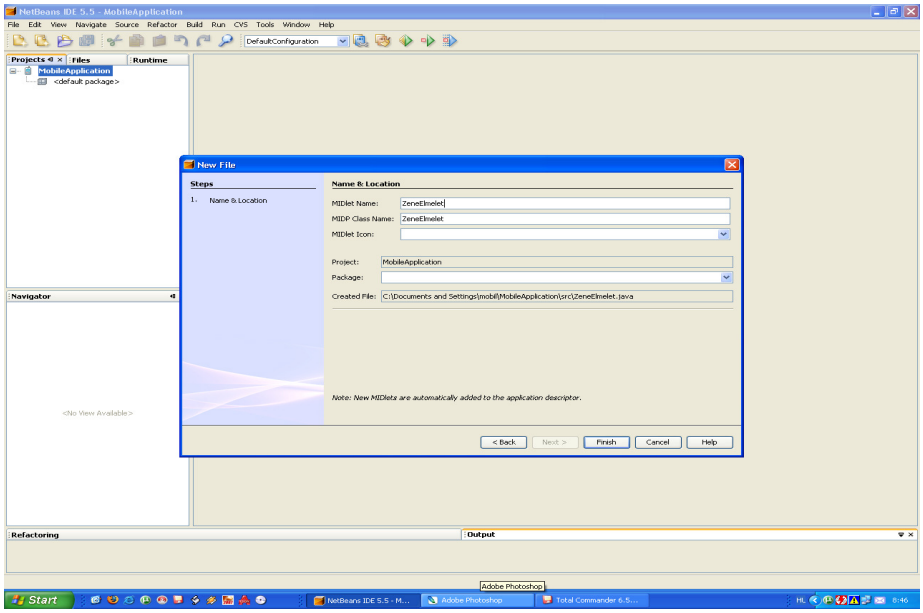
A Next gomb megnyomása után, be kell állítanunk a program által használt mobiltelefonszimulátort. Jelen esetben ez, a beépített, J2ME Wireless Tolkit 2.2 platform lesz. Mivel Nokia 3650-es telefonra kívánjuk majd telepíteni kész alkalmazásunkat, a profilok közül a „MIDP-1.0”-t kell választanunk, mert csak így fog működni a telefonunkon.



A finish gombbal létrehozzuk projektünket.

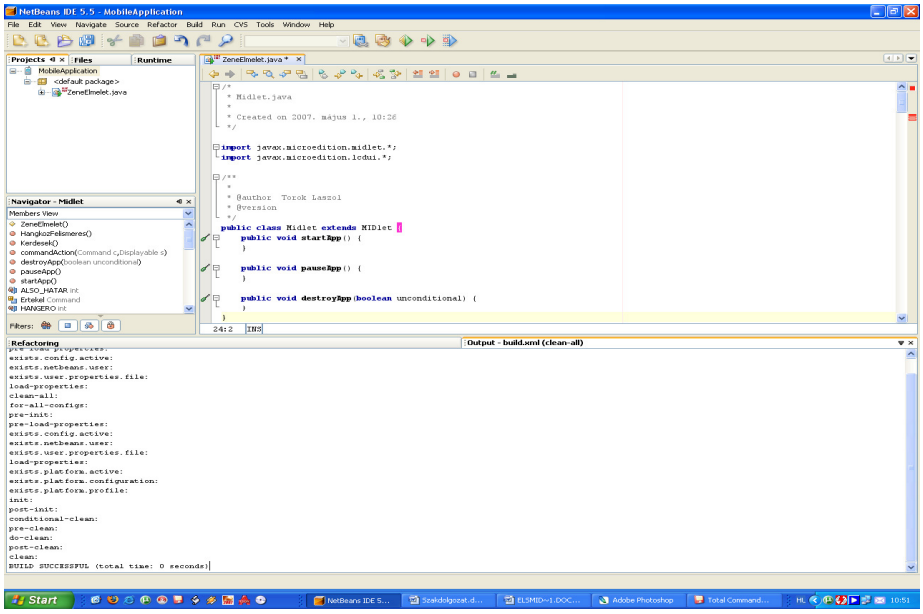
A projekt már megvan, most következik a MIDletünk létrehozása. A projektnéven az egér jobb gombjával kattintva, az új elem hozzáadását választva, a MIDlet soron kattintunk.

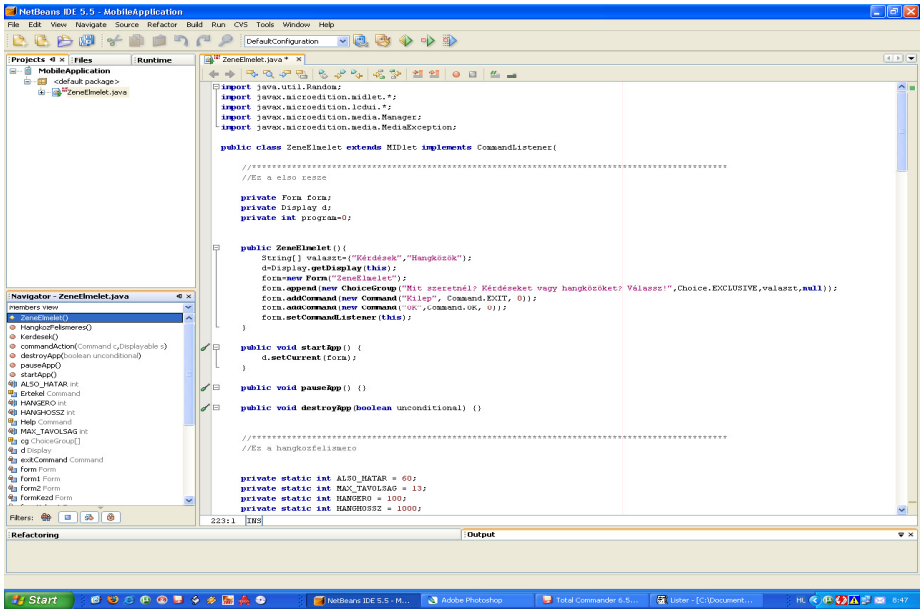
Az ekkor megnyíló panelen meg kell adnunk a MIDlet nevét. Mi a ZeneElmelet nevet adjuk meg MIDlet-névként, valamint a MIDP osztály neve is ez lesz.



A finish gomb megnyomásával kezdődhet a munka.

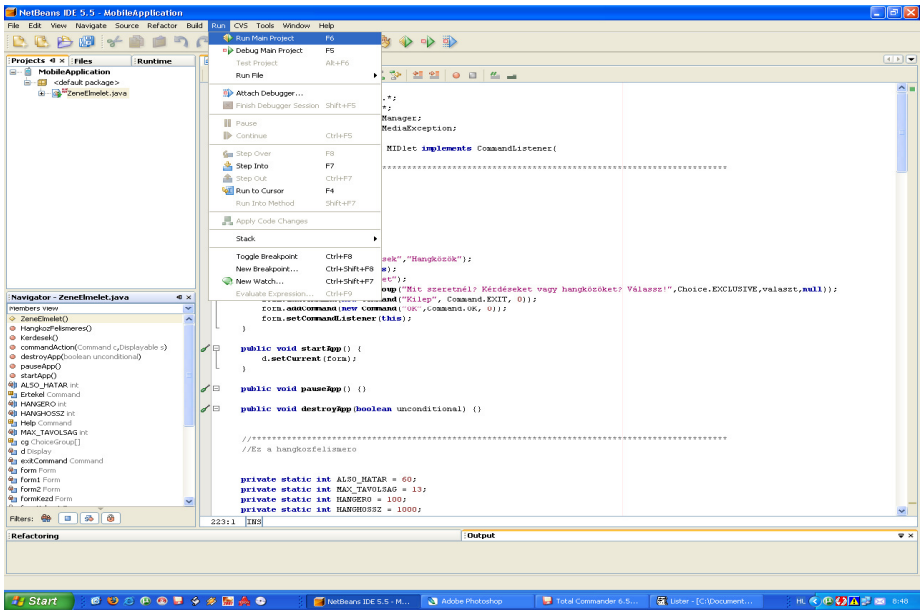
Mivel programunk a MIDlet osztályt terjeszti ki, a leszármazottaknál kötelező implementálni a „StartApp()”,”PauseApp()”, és a ”DestroyApp()” metódusokat. Ezeket a Netbeans automatikusan „beírja” forrásszövegünkbe.

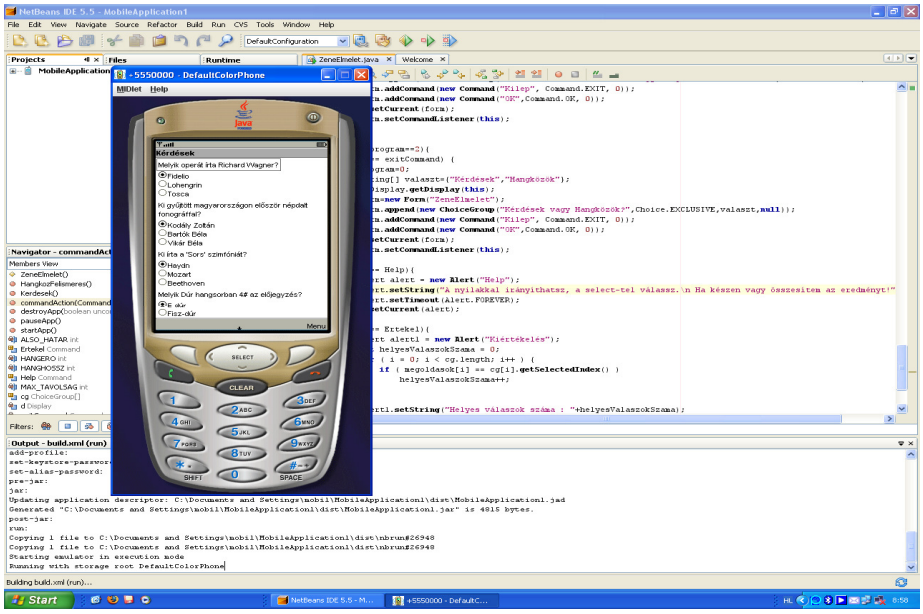




Ha elkészültünk, a „run” legördülő menü „run” main projekt” sorát választva vagy az „F6” funkcióbillentyű megnyomásával, megtekinthetjük, hogy mit is „alkottunk”.

Ekkor a beépített szimulátor elindul, és kipróbálhatjuk Midletünket.





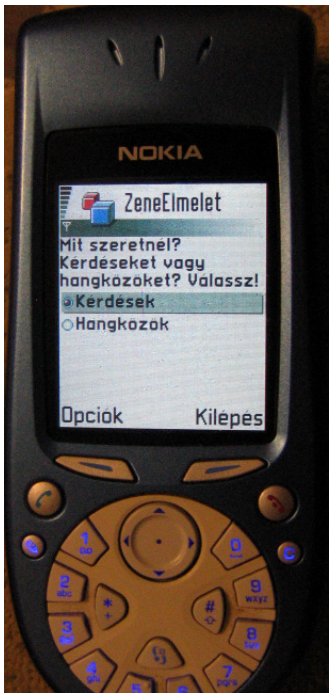
A program működése:

Telepítés Nokia 3650 telefonra

A jar fájlt bluetouth kapcsolaton keresztül átküldjük a telefonra.

A telefon jelzi, hogy a bluetouth kapcsolaton keresztül üzenet érkezett. Az üzenet megnyitásához készülékünk megkérdezi, hogy telepíted–e az alkalmazást. Telepítés után az Alkalmazások mappában megjelenik Midletünk MobileApplication néven. Innen indíthatjuk el.

Indítás után a nyitóképernyőt látjuk. Középen midletünk neve áll: ZeneElmélet. Alatta felirat: Mit szeretnél? Kérdéseket vagy hangközöket? Válassz! Alatta két választható elem: kérdések Hangközök

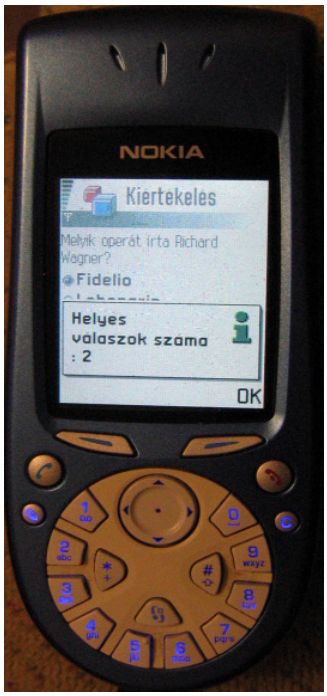


A le és fel nyilakkal navigálhatunk és a „select” gomb megnyomásával kiválaszthatjuk a kívánt elemet (csak az egyiket választhatjuk). Kiválasztom a kérdéseket az OK gomb megnyomásával léphetünk tovább vagy a kilépés gombbal kiléphetünk a programból. Most az OK – val továbblépünk. Ekkor a Kérdések programrész képernyőjét látjuk.

A kijelző tetején programrészünk neve látható: Kérdések.

A program kérdést tesz fel és minden kérdéshez 3 – 3 választ ajánl fel. A felajánlott válaszok közül egyet – egyet kell választanunk. A le – fel nyilakkal navigálhatunk a kérdések és a felajánlott válaszok között, majd a select gombbal kijelöljük a kívánt válaszokat. A kijelző alján látható az Opciók felirat. A hozzá tartozó gomb megnyomásakor előgördül egy menü, amely a következő lehetőségeket biztosítja: Exit Értékek Help Kilép

A Help-et választva egy súgóüzenetet kapunk, ami elmagyarázza a kérdéses programrészünk használatát. Az OK gomb megnyomásakor a súgó ablak eltűnik, és ott folytathatjuk a kérdések megválaszolását, ahol abbahagytuk. Amikor úgy gondoljuk, hogy kiválasztottuk a megfelelő válaszokat, megnyomjuk az opció gombot és az előgördülő menüből kiválasztjuk az „Ertekel” lehetőséget. Ehhez egy ablak jelenik meg, amelyből megtudhatjuk, hogy hány helyes válaszunk volt.



Az OK gomb megnyomásával visszatérhetünk a kérdésekhez, és esetleg folytathatjuk a válaszok kiválasztását.

Ha át akarunk térni a hangközrészre az opciók közül az exitet kell választanunk. Ekkor visszatérünk a nyitóképernyőre. Itt a hangközöket kiválasztva megnyomjuk az OK gombot. Ekkor egy üdvözlő képet látunk: hangközfelismerés névvel.

Üdvözöllek a hangközfelismerő programban. Az első hang lejátszásához nyomj OK – t!

Megnyomjuk az OK – t és halljuk az első hangot. A hang megszólalása után a második képernyőt látjuk, melyen a következő felirat áll: A második hang lejátszásához nyomj OK – t.

Megnyomjuk az OK gombot és meghallgatjuk a második hangot. Ekkor láthatjuk a harmadik képernyőt melyen a „Milyen hangközt hallottál?” kérdés áll valamint láthatunk B választható elemet, amelyek közül ki kell választani a hallott hangoknak megfelelőt.

A kiválasztás után OK – gombot nyomva a negyedik képernyőt látjuk, melyen „A válasz helyes!” vagy „Rossz válasz! A helyes válasz: …..” felirat olvasható, attól függően, hogy miképpen választottunk a válaszlehetőségek közül, valamint választhatunk az újra, és a kilép gombok között.

Ha az újra lehetőséget választjuk, akkor ismét próbálkozhatunk a hangközfelismeréssel. Ha kilépünk, akkor a legelső nyitóképernyőre jutunk, ahol ismét választhatunk a Kérdések és hangközök között, vagy a kilépés gombbal kiléphetünk a programból.

Összegzés:

A bevezetésben leírt célokat sikerült megvalósítani. A szakkörre járó gyerekek a program elkészítése közben megismerkedtek a java nyelv történetével, sajátosságaival, összehasonlítva a már ismert programozási nyelvekkel.

Áttekintettük az objektumorientált programozást a java nyelven keresztül. Elsajátították a mobilprogramozás alapjait, megismerkedtek a használatos fogalmakkal. A program elkészítése közben, mindezek mellett zeneelméleti ismeretekre is szert tettek.

Bízom benne, hogy a zene és az informatika ilyen módon történő összekapcsolása ösztönzően hat a gyerekek tanulmányaira a jövőben is, mind a zene mind az informatika terén.

Irodalom:

[1] Móricz, Attila, Java Alapismeretek. Budapest, LSI Oktatóközpont, 1997.

[2] Móricz, Attila, Java programozási nyelv I-II. Budapest, LSI Oktatóközpont, 1997.

[3] Kiss István: a java programozási nyelv

[4] KVM White Paper; Sun Microsystems, Inc.
http://java.sun.com/products/cldc/wp/KVMwp.pdf

[5] A Sun Java hivatalos oldala
http://java.sun.com/javame/index.jsp

[6] Paller Gábor: MIDletek
http://pallergabor.uw.hu/hu/java-app/MIDletek.html

[7] A Netbeans letöltése:
http://java.sun.com/javase/downloads/netbeans.html

[8] A Netbeans Mobility platform letöltése:
http://www.netbeans.info/downloads/index.php?rs=11&p=4

[9] A Java ME2 technológia dokumentációja:
http://java.sun.com/javame/reference/apis.jsp
http://java.sun.com/javame/reference/apis/jsr030/
http://java.sun.com/javame/reference/apis/jsr139/
http://java.sun.com/javame/reference/apis/jsr037/
http://java.sun.com/javame/reference/apis/jsr118/