**Debreceni Egyetem**

**Informatika Kar**

# Mobil alkalmazásfejlesztő platformok összehasonlítása

**Témavezető:**
**Bátfai Norbert**
**tanársegéd**

**Készítette:**
**Zelenák Zoltán**
**programtervező informatikus**

**Debrecen**
**2010**

# Tartalomjegyzék

# 1. Bevezetés

## 1.1 Előszó

Témám címe rejtélyes lehet az olvasónak. Be kell valljam, számomra is az volt, vagy legalábbis, teljesen más elképzelésem volt a mobil alkalmazásokról azelőtt, mielőtt témám megírására vállalkoztam. Még ma is sokan, ha meghallják ezt a szót: mobil, azonnal a mobiltelefonokra gondolnak. De be kell látnunk, hogy a mobil eszközök, mint kategória sokkal szélesebb annál, minthogy egyedül ezen eszközökre szorítsuk. Mikor gyerekkoromban egy Tetris -szel játszottam, vagy épp az akkoriban divatos Tamagochi-kat láttam, meg nem fordult a fejemben, hogy azok is ebbe a kategóriába tartoznának. Vagy megemlíthetnénk a videokamerákat, és a digitális fényképezőgépeket, de idetartoznak a különböző média megjelenítésre, lejátszásra képes eszközök is.

Példáim alapján már kezdhet megfogalmazódni bennünk a gondolat, hogy mik is azok a mobil eszközök. Ha egy definíció után néznénk, sajnos nem találnánk rá egyértelmű választ. Egy egyszerű megfogalmazás, ha olyan számítógépről beszélünk, mely hordozhatónak és hordozás közben kényelmesen használhatónak lett tervezve. A kényelmesség fontos, hisz például egy noteszgép is hordozható, s még egy autó utasaként is igen jól tudjuk alkalmazni, de egy folyosón, ahol nem tudjuk lehelyezni sehová, már igencsak bajba juthatunk az alkalmazásával. Alkalmazásuk pedig széleskörű. Ez értelmezhető szerepkörileg, földrajzilag vagy akár időben is.

Láthatjuk, hogy a mobil eszközökről alkotott kép cseppet sem egységes. És ez így jelenik meg a gyakorlatban is, még pedig a platformok formájában. De hogy ne legyen olyan egyszerű a helyzet, mindjárt 3 szinten is. Mobil eszközök esetén beszélhetünk ugyanis hardver platformról, szoftver platformról és fejlesztői platform. A hardver platform azt a mobil eszközt mint tárgyat jelenti, amelyre fejlesztünk, a szoftver platform a keretrendszer amiben futni fog a programunk, a fejlesztői platform pedig a programozási nyelv és a hozzá tartozó lehetséges fejlesztői környezeteket jelenti.

Mikor egy mobil alkalmazás fejlesztésébe fogunk mindhármat mérlegelnünk kell. Szerencsésebb esetben akár még az egyik szinten megválasztott platform adhatja a másik kettőt is, de a legtöbb esetben azzal találkozunk, hogy több platformra is el kell készítenünk ugyan azt a

programot. Ilyenkor jelentős, hogy mennyire hordozható maga a kód amit írtunk. Ez egyrészt függ a platformtól is, de a fejlesztőnek is meg kell állapítani, hogy mely része az alkalmazásnak az, amelynél lehetséges és hasznos az, hogy ugyan úgy nézzen ki több platformon is, vagy melyek azok amelyeknél ugyan lehetséges lenne a hordozhatóság megvalósítása, ugyanakkor olyan környezeti eltérés ( környezet alatt pedig értendő a hardveres és a szoftveres platform) létezik, mely miatt a kód átírása javallott. Lehetséges ugyanis, hogy az egyik hardver platform olyan előnnyel rendelkezik, amelyikkel a másik nem, s bár időt és pénzt spórolnánk egy általánosabb megoldással, de ezáltal egy jobb terméktől esnénk el.

Szakdolgozatom ezen kérdéseket boncolgatja. Egy általános rálátást biztosítani jelen korunk platformjaira, és példákon keresztül részletesebben is tárgyalva mutatnám be, hogy jelennek meg a különbségek és hasonlóságok a szoftverfejlesztésben.

### 1.2 Történeti áttekintés

Mielőtt azonban a lényegi tárgyalásra térnénk, nem árt megismerkednünk valamelyest a mobil eszközök történelmével. Sokan ezt a mobiltelefónia hajnalával hozzák összefüggésbe, de azon eszközök inkább mint távközlési eszközök voltak jelentősek, másra mint telefonálásra nem voltak használhatóak. Az úgy nevezett 0 -ik generációs mobiltelefonok már a második világháború után megjelentek, mely csak annyiban tért el a korábbi rádiótelefonos megoldásoktól, hogy egy központi szolgáltatón keresztül lehetett elérni a másik félt. A mai mobiltelefonálás alapját, a cella alapú telefonálásra a 70-es évekig várni kellett. A Motorola -nál fejlesztették ki az első valóban kézben hordható telefont még 1973-ban, de a technológia forgalomba kerülésére 1979-ig várni kellett. Ekkor vált bárki számára elérhetővé Japánban az első cella-hálózat.

Ezt megelőzőleg azonban már a Mattel is piacra dobta az első hordozható konzolját, az Auto Race-t, 1977 -ben. Csak úgy, mint a korabeli mobil eszközök, megvalósítása teljesen áramköri szinten történt, nem utólagos programozás útján. További előrelépést ugyancsak a játékipar hozott, mikor 1980-ban a Nintendo előállt a Game&Watch szériájával. Ennek különböző klónjai Kelet-Európában is népszerűek voltak. Itt már ROM -ban tárolt programról beszélünk, s a kezdetleges grafikus megjelenítés a kijelző borításán látható képek egészítették ki.

A 80-as évek csúcskategóriát képviselő mobil eszközei azonban a grafikus számológépek voltak. 1985 -ben jelent meg a Casio fx-7000G típusjelű grafikus számológépe, az első kategóriájából. A korában nagyméretűnek számító kijelző, és az, hogy a felhasználó írhatott rá programokat, melyek közül egyszerre 10-et tudott 422 byte nagyságú memóriájában tárolni, előrevetítette, hogy mi várható a mobil eszközök piacán a jövőben.

A következő fontos állomás az 1989-es év. Ekkor jelent meg a piacon a Nintendo a GameBoy nevű termékével. A GameBoy hordozható játékkonzol, csakúgy mint a Game&Watch sorozat tagjai, azonban a játékok már adathordozón(speciális cartride -on) voltak kaphatóak. A kijelző felbontása 160×144 pixel, a memória mérete pedig 8 kbyte volt. Látható, ez hatalmas ugrás a grafikus számológépekhez képest. A gép rendelkezett speciális, csak a grafikus megjelenítést szolgáló hardveres megvalósításokkal. Ilyenek például a hardveres háttér mozgatás, és az úgynevezett hardver sprite-ok kezelése. A játékok assemblyben íródtak, speciális fejlesztői rendszer volt szükséges hozzájuk, amelyeket a Nintendo -tól lehetett speciális szerződésen keresztül megszerezni, és csupán a Nintendo engedélyével lehetett megjelentetni rá bármilyen játékot is. Így a Nintendo egyfajta minőségi kontrollal rendelkezhetett a megjelent programok felett, míg a fejlesztők egy átlátható rendszert kaptak a programjaik terjesztéséhez. Ez egyébként bevett szokás volt az asztali konzolok piacán minden gyártó részéről már a 70-es évek óta. A GameBoy bemeneti perifériája korlátozott volt. Összesen egy 8-irányú pad, és 4 egyéb gomb, melyek közül 2 általános célú és egy kiemelt start és select gomb, volt a gépen megtalálható. Ez azt jelentette, hogy a fejlesztőknek az irányítás egyszerűségével is kalkulálnia kellett a fejlesztés folyamán. Ez igaz a későbbi mobil eszközök majd mindegyikére, játékra való felhasználásnál körülbelül hasonló gombszámra számíthatunk, de bizonyos különbségekre majd a megfelelő résznél kitérek még. Nem utolsó sorban pedig tudni kell, hogy a GameBoy volt az a készülék, amely a mobil eszközöket nagy nyilvánosság elé hozta.A GameBoyból 118 millió kelt el világszerte, a Tetris GameBoy-os változatából pedig mintegy 35 millió darab talált gazdára.

A GameBoy megjelenése után 4 évvel mutatta be az Apple az első PDA-kat, egy évben az SMSezés szélesebb körben való megjelentetésével. Utóbbi kiemelten fontos a mobil eszközökre fejlesztett programok tekintetében, hiszen ennek megvalósítására születtek az első mobiltelefonos alkalmazások. Az első PDA a MessagePad 100 volt. A ceruza mint elsődleges információbevitel,

a speciális mobil operációs rendszer használata mindmáig jellemző a PDA-k többségére, azonban a kezdeti modellekben még nem volt lehetőség a számítógéppel való szinkronizációra. Ennek, rövid üzemidejének és annak, hogy a külső fejlesztők nem láttak benne üzletet, köszönhetően a MessagePad nem lett nagy siker.

A 90es évek második felében váltak széleskörben elterjedté a PDAk. Az időszak vezető cége a Palm volt, melynek egyaránt szoftveres és hardveres megoldása is uralkodó volt. A cég által fejlesztett Palm OS-t több más mobil eszközben is alkalmazták, többek közt még karóra és vonalkód leolvasó is lett ezen szoftverrel felszerelve. A Microsoft 1996 -ban jelentette be a Windows CE 1.0 -ás verzióját. A Windows CE a Microsoft beágyazott rendszerekre szánt komponens alapú operációs rendszere, amely azt jelenti, hogy a kiválasztott célplatformnak megfelelően az operációs rendszer csupán bizonyos moduljai lesznek azon platformon elérhetőek. Egy ilyen megvalósítása a rendszernek a Windows Mobile. A Microsoft mobil platformon azonban egészen az ezredfordulóig nem tudott előretörni.

A PDA-k elterjedésével egyidejűleg jelentek meg az első smartphone-ok is. 1996 -ban került kereskedelmi forgalomba a Nokia cég 9000 Communicator nevű okostelefonja. Operációs rendszerrel rendelkezett, és internet elérésre is képes volt, de ezt csupán GSM rendszeren keresztül. Ezenkívül egy 640×200 -as kijelzővel rendelkezett, amely még a korabeli PDA -k közt is nagy méretűnek számított. Érintő képernyővel pedig nem rendelkezett, amely még sokáig jellemző volt az okostelefonokra.

1998-ban készült el a WAP 1.0-s szabvány, amellyel a mobil-internet szabványosítását tűzték ki célul. A mobiltelefonok egy WAP-böngészőn keresztül érhetik el a WML-ben (Wireless MarkUp Language) íródott honlapokat. Ezek az oldalak specifikusan a mobiltelefonokra vannak optimalizálva, ide értve az ekkortájt törpe méretű kijelzőt, korlátozott felbontást és színmegjelenítést. A WAP elterjedése azonban lassú volt. Főleg azért, mert az ekkortájt elérhető mobiltelefonokon a felhasználói élmény, amelyet a WAP szolgáltatott meglehetősen szegényes volt.

2000 után egymást hajtva váltak népszerűbbé a mobil eszközök és lett nagyobb a teljesítményük. A Nokia 3210-es mobiltelefon egyfajta néptelefonná vált. Ez a telefon már rendelkezett beépített játékokkal, csengőhang szerkesztéssel, grafikus megjelenítéssel, bár a

kijelző még csupán 2 színű volt. Ez a készülék tette a legtöbb ember számára ismertté az úgy nevezett kígyós játékot. Az évtized elején jelent meg a Gameboy Advance, amely a fő mobil játékfejlesztési platform volt az évtized első felében. A PalmOS és Windows CE után a Symbian OS volt a következő általános elterjedtségű operációs rendszer, amely a Nokia támogatásának köszönhetően hamarosan a vezető mobil operációs rendszerré vált. Ebben az időben szorult vissza a PalmOS és jelent meg a Windows Mobile változata a Windows CE-nek. S mindenek tetejében a RIM is ekkor szállt be a mobil eszközök piacára az első BlackBerry-vel. A sok különböző platform között hidat jelentett a Java mobil eszközökre készített szabványa, amely pár év alatt a mobil eszközök java részén elérhető volt.

Az évtized elején még csupán a felsőbb kategóriás készülékeket illette meg a színes kijelző, a telefonok közti adatátvitel, a telepíthető programok és a hálózati kommunikáció. Azonban az először még csupán luxusnak számító szolgáltatások az alsóbb kategóriákban is elterjedtek. A mobiltelefonok és a PDA-k közti először éles határvonal lassanként elhalványult. Egyrészt megjelentek telefonálásra is alkalmas PDA-k, másik oldalon viszont a mobiltelefonok is felruházódtak addig inkább a PDA-kra jellemző vonásokkal. Emellett az ekkortájt terjedőben lévő friss mobil eszközökkel is egybe integrálódtak, mint amelyek a digitális kamerák és multimédia lejátszók.

Míg a 2000 utáni időszakot a Symbian piaci dominanciája jellemezte, addig az évtized második felében a piac átrendeződésének és új igények kialakulásának lehetünk tanúi. Megfigyelhető, hogy a specializált eszközöket, mint amelyek a digitális kamerák, médialejátszók, GPS-ek nem képesek kiszorítani az új egyre több funkcióval ellátott eszközök. A multifunkciós és a specializált eszközök olyan szinten férnek meg egymás mellett, hogy míg az egyik a minőséget, a másik a kompaktságot helyezi előtérbe. A hagyományos PDA-k és okostelefonok eltűnnek, helyettük a modern okostelefonok válnak uralkodóvá. Fontos állomás ebben az Apple forradalma, amelyet az iPhone-nal indított el. Az Apple forradalma központjában a felhasználói interakció állt. Ugyanis míg a korábbi eszközök inkább mobiltelefonnak látszóan, tehát gombokon keresztül próbálták lehetővé tenni az információbevitelt, avagy inkább PDAnak látszóan, stylus használatával, az asztali gépeket mimikáló felhasználói felülettel, addig az Apple mindezek mellőzően egy ikonalapú, gyors, reszponzív rendszert készített. A platformra készült

programok ráadásul zártan, az iStore-on keresztül érhetőek el, amely jelentősen megkönnyíti az új szoftverek beszerzését. Ez gátolja az illegális szoftverek terjedését, és kényelmes mind a felhasználó, mind a fejlesztő számára. Az Apple-lel párhuzamosan a RIM volt még, amely képes volt jelentős térnyerésre, köszönhetően annak, hogy míg az Apple újításai az átlagfelhasználók közt váltak népszerűvé, addig a RIM az üzleti felhasználásra tervezett készülékekkel állt elő. A RIM készülékei képesek szinkronizációra a jelentősebb csoportmunka szoftverekkel, ezenkívül a vállalat belső levelezési rendszerével való integrációra. Az iPhone és a Blackberry térnyerése a Symbian-t és a Windows Mobile-t érintette kényesen. A Symbian bár még mindig piacvezető, érezhetően sürgős ráncfelvarrásra szorul, amelyen jelen pillanatban is igyekszik a Nokia. A Windows Mobile szerepe majdnem teljesen eljelentéktelenedett. A Microsoft szintén dolgozik a jelentős újításokkal rendelkező új verzión.

A Java technológia ezen évek alatt főleg az alsóbb kategóriás telefonokon hódított, bár sok különböző fejlesztés jelent meg, igazából a Java-s játékok piaca lett az amellyel jelentős sikereket tudott elérni. A komolyabb hardverplatformokon a natív megoldások az elsődlegesek. A Sun-os Java direktívák mellett jelent meg a Google az Android-dal, mely több ponton szemben megy a Sun-os szabvánnyal. Az Android Linux-os alapokra épül, míg az alkalmazások rá Java-ban íródnak és egy speciális virtuális gépen futnak. Az Android-os piac még most van kialakulóban, jövője még igen csak kérdőjeles. Az Android azonban nem az egyetlen platform, amely Linux-os alapokra épül. A Nokia a Symbian OS új, modern verziója hiányában és a mellett jelent meg a Maemora, amely egy Debian Linux disztribúcióra épülő rendszer erősen megtámogatva a szintén Nokia által fejlesztett Qt multiplatform GUI keretrendszerrel.

Láthatjuk, hogy a 10-es évek elejéhez érve a mobil eszközök palettája hihetetlenül széles, és kaotikus. Fontos látni, hogy jelenleg még nem alakultak ki egyértelmű piaci helyzetek, jelennek meg új termékcsoportok, régiek eltűnnek, s ugyan ez vonatkozik a piaci szereplőkre is. Figyelembe kell venni, hogy a piac méretei hatalmasak, az International Telecommunication Union szerint 2009 végére 4,6 milliárd a mobiltelefon-előfizetések száma, s ez még nem foglalja magába az egyéb mobil eszközök számát. Azonban a felhasználói igények nagyon különbözőek, s így mind a hardver és szoftverfejlesztőknek is tisztában kell lennie ezen igényekkel.

# 2. Hardverek

## 2.1 A mobil hardverek természetéről

Amikor alkalmazás fejlesztésbe fogunk, figyelembe kell vennünk a célhardver által nyújtott lehetőségeket. Látnunk kell ugyanis, hogy a mobil eszközök között különösen nagy különbségek vannak hardveres téren. Például míg egy asztali számítógépnél elvárhatjuk, hogy az egér és a billentyűzet az általános bemeneti periféria, addig a mobil eszközöknél ilyen általános jellegű dologról nem beszélhetünk. Hogy azonban még is legyen valamilyen támpontunk, a mobil eszközök csoportosíthatóak. A legáltalánosabb csoportosítás azonban ezen eszközök funkcióján alapul. Ez az, amelyet én is használni fogok a következőkben. Ez a csoportosítás azonban még így is hagy kívánni valót maga után, főleg a mobiltelefonok esetében, melyek a legszélesebb palettát mutatják hardverek tekintetében.

A mobil eszközökre való alkalmazásfejlesztésnél ezen kívül meg kell különböztetnünk 3 hozzáállást a szoftverek disztribúcióját illetően. Az első esetben a hardver gyártó látja el a szükséges szoftverekkel a terméket még a termék piacra kerülése előtt. Általában az ilyen eszközök utólag nem módosíthatóak, esetleg a gyártó kínálhat fel frissítést az eszköz firmware-ére. A második esetben a termék ellátható szoftverekkel utólag is, de a szoftver csak a hardver gyártóval kötött szerződés után fejleszthető rá, ezeket nevezzük zárt rendszereknek. A harmadik esetben nincs korlátja a szoftver fejlesztésének. Ebben a fejezetben csupán az utolsó két kategóriával foglalkozom részletesebben.

## 2.1 Hordozható konzolok

A hordozható konzolok, csak úgy, mint nagyobb testvéreik, elsősorban játékok futtatására kifejlesztett hardverek. Ebből következik, hogy a grafikus megjelenítés kulcsfontosságú ezen eszközök között. Azonban, szintén nagyobb testvéreikhez hasonlóan, ennél már jóval szélesebb körben felhasználhatóak. A piac vezető szereplői napjainkban a Nintendo DS és a Playstation Portable. Mindkét eszközre a fejlesztéshez szükséges a gyártó céggel való megállapodás. Emellett léteznek nyílt eszközök is, de ezek részesedése a piacból elhanyagolható.

Egy átlagos modern hordozható konzol rendelkezik egy viszonylag nagy méretű kijelzővel, és 3 dimenziós grafikus megjelenítésre képes hardverrel. Ezen kívül az adatbevitel egy 8 irányú paddal és legalább 6 általános célú gombbal történik. Ma már általánosnak vehető a Wi-Fi kapcsolat is, és valamilyen háttértároló megléte is. A programok főleg valamilyen külső adathordozón tárolódnak, de elterjedőben van a digitális disztribúció. Az eszközök operációs rendszerrel nem rendelkeznek, egy firmware töltődik be induláskor, amely teljes kontrollt ad az éppen futó alkalmazásnak. Ezen általános jellemzőktől azonban elég jelentős eltérések is lehetnek. A Nintendo DS például 2 kisebb kijelzőt használ egy nagyobb helyett, illetve érintőképernyővel is rendelkezik.

A zárt konzolokon a szoftvererek fejlesztése a hardvergyártótól kapott fejlesztőrendszereken történik. Ezek a fejlesztő rendszerek egy speciális fejlesztői hardvert és egy szoftveres környezetet foglalnak magukba. A fejlesztés operációs rendszer hiányában jóval hardver közelibb, mint más rendszereken, ráadásul az eszközök általában a megjelenítéssel kapcsolatban speciális instrukciókkal is rendelkeznek. Ezek következtében a kódok nagy része nem hordozható a hordozható konzolok közt.

Szoftverellátottság tekintetében, pedig mindenképp fontos megemlíteni, hogy ezen eszközök is ma már multifunkciósak. Ugyan úgy alkalmasak filmnézésre, web böngészésre vagy éppen határidő napló kezelésére, mint más mobil eszközök. És az ilyen jellegű szoftverek nem csupán a hardver gyártótól származnak, hanem a külső fejlesztő cégektől is megjelennek. Jellemző lehet még az előző generáció szoftvereiknek használhatósága vagy hardveres vagy szoftveres alapokon. Első esetben az újabb generáció tartalmazza az előző generáció hardverelemeit, így ezen gépek natívan képesek futtatni az előző generáció játékait, míg a második esetben szoftveres hardver-emulációt alkalmaznak.

### 2.3 PDAk és okostelefonok

A mobiltelefonok csúcskategóriáját ma már kénytelenek vagyunk egybevenni a PDAk világával. A régebben csak PDAkra koncentráló gyártók is ma már szinte csak mobil telefonálásra is alkalmas készülékekkel állnak elő, illetve a mobiltelefongyártók csúcskategóriás készülékei is teljesen funkcionális PDAk. Ezeket pedig összességében okostelefonoknak

nevezzük.

Ezen kategóriának 2 fő jellemzője van: rendelkeznek valamilyen mobil operációs rendszerrel és a fő beviteli eszköz az érintőképernyő. Általában az operációs rendszertől függ, hogy a programok szabadon fejleszthetőek-e vagy sem. A fejlesztői környezet is leginkább az operációs rendszertől függ. Ennek köszönhetően már több különböző gépen is futhat ugyan az a kód, de a gépek különböző képességei miatt a végeredmény közel sem ugyan az minden típuson. A kompatibilitás megoldásáért a használt API felel, amely a géptípusonkénti különbségeket absztrahálja. A legtöbb API garantálja az alacsony szintű hozzáférést is, és a fejlesztő feladata eldönteni, hogy az előre kész elemeket, vagy a saját megvalósítását használja.

Az okostelefonok fontos jellemzője, hogy nagyon sok technológiát integrálnak egybe. A legtöbb okostelefon rendelkezik kamerával, ismer több vezeték nélküli adattovábbítási szabványt, távközlési szabványokat, rendelkezhet GPS-el és giroszkóppal. Tőlünk nyugatabbra már az okostelefonoknál is elvárható a majdnem folytonos netkapcsolat. Ez az alkalmazások megvalósításában is megjelenik, ugyanis egyre több alkalmazás használja ki ezen lehetőségeket.

A bevitel azonban nem olyan egyértelmű. Némely okostelefonok rendelkeznek QWERTY billentyűzettel, mások csupán minimális számú hardveres megoldással, és soft gombbokal rendelkeznek. Ez a két megoldás jelentősen befolyásolja az információbevitelt. Saját tapasztalataim is azt mutatják, hogy a hardveres gombokra tervezett alkalmazások, főleg ha teljes képernyőt igénybe veszik majdnem használhatatlanok lesznek egy szoftveres billenytűzettel ellátott telefonon.

Összegezve, az okostelefonok szerepe sokrétű. Egyrészt a munkahelyeken is jól használhatóak, a levelezésen túl is egyre több szerep jut nekik, kiváló eszközei lehetnek a csoportmunkának, másrészt pedig az úgy nevezett „social networking”-ben betöltött szerepe is egyre nő ezen eszközöknek. A felhasználók képesek elérni blogjaikat, twittereiket a telefonról, videóikat feltölteni, és megtekinti, közösségi oldalaikon részt venni. Ez a terület pedig a kisebb fejlesztők számára is rengeteg ötletes alkalmazás elkészítését teszi lehetővé.

### 2.4 Mobiltelefonok

A hagyományos mobiltelefonok általában nem rendelkeznek operációs rendszerrel, csupán

egy firmware fut rajtuk. Ezáltal jóval szűkösebb lehetőséget tárnak a fejlesztők elé. Az, hogy saját alkalmazás írható-e rá, vagy egyáltalán a fájl rendszer használható-e, nagyban függ a készülék gyártójától. Jellemző, hogy ezek a mobiltelefonok nagy többsége képes Java-s alkalmazások futtatására. Ez elméletileg segít abban, hogy az alkalmazások több készüléken is futhassanak, de az összetettebb alkalmazások saját GUI megoldásai jelentősen rontanak annak az esélyén, hogy minden készüléken használható programot kapjunk. Ennek folyamán a mobiltelefonok esetén különösen fontos a készülékenkénti tesztelés.

A programok fejlesztésénél ugyancsak figyelembe kell venni, hogy a kijelző mérete kisebb a fentebb említett mobil eszközöknél. A menük általában egyszerűbbek, a grafikus megjelenítést lehetőleg minimalizálni kell. Mivel az alkalmazáson belül navigálni körülményesebb, ezért erre is figyelemmel kell tekinteni az alkalmazás tervezésekor.

### 2.5 Tablet PC-k és médialejátszók

A tablet PC-k sok esetben kívül állónak tekinthetők a mobil eszközök között. Ezek azok, amelyek speciális notebook-oknak tekinthetők. Egy másik csoport, amelyen valamilyen mobil operációs rendszer fut. Az ilyenek leginkább egy nagyra nőtt PDA-nak tekinthetők. Épp ezért a van egyfajta kompatibilitás a különböző, ámde ugyanazt az operációs rendszert futtató tablet PC-k és okostelefonok közt. A tapasztalat azonban azt mutatja, hogy a különböző kijelző méret jelentősen megnehezítheti az alkalmazások alkalmazhatóságát.

A médialejátszók egy időben nagyon egyszerű eszközök voltak. Azonban mára ezek is sokféle funkcióra tettek szert. Először csak a különböző médiaformátumok lejátszási képességei kerültek be, de mára a komolyabb médialejátszók képesek alkalmazások futtatására is. Ez általában valamilyen mobil operációs rendszer segítségével valósul meg. Az ilyen komolyabb eszközökről is elmondható azonban, hogy általában csak érintőképernyővel rendelkeznek és ezen kívül meglehetősen kevés gomb található rajtuk.

Fel tehető a kérdés, hogy mi a különbség egy médialejátszó és egy PDA között. Ez a kérdés természetesen csupán az operációs rendszerrel ellátott verzióknál igaz, hisz alap esetben a médialejátszók jóval elmaradnak a PDA-khoz képest. Azonban a fejlettebb modelleknél tulajdonképp nem mindig lehet megkülönböztetni a kettőt. Van, amikor a médialejátszó speciális

szoftverrel van ellátva, míg a PDA-ra egy másik megoldást kínál a gyártó. Ez konkréten a Microsoft Zune HD-jának esete, melynek operációs rendszeréből az éppen készülő Windows Mobile 7 elemei köszönnek vissza. Ezzel szemben az Apple iPhone és az iTouch operációs rendszere megegyezik, a gyakorlati különbség bizonyos extra elemek hiányában mutatkoznak meg.

### 2.6 Egyéb eszközök

Az eddig említett hardvereken kívül megemlítendő még sok kisebb eszköz, melyek főképp célspecifikusak, és nem nyitottak saját alkalmazások befogadására. Mint a fejezet bevezetőjében említetettem, ezekkel nem foglalkozom bővebben, de a fentebbi kategóriák kontextusában egy két dolgot mégis meg kell említenünk.

Az első az e-book reader-ök elterjedése volna. Ezek elsősorban szövegmegjelenítő eszközök, melyek könyvolvasásra használhatók. Azért emelném ki őket, mert van létjogosultságuk az okostelefonok és a tablet PC-k világában. Ezen eszközök ugyanis valószínűleg mindig szembarátabb és energiatakarékosabb technológiát fognak használni, mint a sokkal sokoldalúbb felhasználást megkövetelő előbb említett eszközök. Hasonló a helyzet a videokamerákkal és a digitális fényképezőgépekkel. Ugyan már az okostelefonok is meglehetősen erősek ezen a téren, könnyű belátni, hogy szükségszerű kompaktságuk miatt, nem lesznek képesek kiszorítani ezen termékeket a piacról. Ez azonban nem mondható el a navigációs segédeszközökről, melyek képességeiket tekintve most sem számítanak előremutatónak az okostelefonok képességeihez mérten, így e platform jövője előtt egy nagy kérdőjel áll.

### 2.7 A hardverekről összegzésképp

Összességében tehát 4 nagyobb kategória volna, amelyre a külső fejlesztések nagy része koncentrálódik. Ezen kategóriák közt azonban már most látni lehet, hogy vannak átfedések, és a hordozhatóságot nem feltétlen a hardver, hanem a rajta futtatott operációs rendszer határozza meg. Érezhető az is, hogy maga a logika hasonlóképp írható le a legtöbb gépen. Jelen korunkban leginkább az alsó kategóriás telefonok azok, amelyek még korlátok közé szoríthatják a fejlesztőket, a nagyobb rendszereknél inkább a megfelelő kihasználás a kérdéses. Amik

problémát jelentenek, azok leginkább a megjelenítésből és a felhasználó általi interakcióból fakadnak. Ezek azok az elemek amelyek a legtöbb munkát igényelhetik géptípusok közt, de mint látni fogjuk, ennek egységesítésére is vannak törekvések. Azonban a hardvereket vizsgálva egyértelműen nem jelenik meg az, amely a következő fejezetben már inkább kikristályosodik, hogy a fejlődés egyszerre halad a nyíltabb és a zártabb rendszerek felé, és a két oldal közül még nem lehet látni, melyik lesz a befutó.

# 3. Operációs rendszerek és multiplatform megoldások

## 3.1 Operációs rendszerek mobil eszközökön

Az előző fejezetből láthattuk, hogy nagyon sok különböző hardverrel van dolgunk a mobil eszközök közt. A külső, tehát nem a gyártótól származó, fejlesztések pedig főleg a fentebb említett 4 kategóriára koncentrálódnak. Ebből 1 kategória az, a PDA-ké és az okostelefonoké, amellyel szemben alapvető elvárás, hogy rendelkezzen operációs rendszerrel.

A modern mobil operációs rendszerek mind külsőleg, mind belsőleg nagyjából megegyeznek asztali társaikkal. Hogy a jóval kisebb teljesítményű hardvereken fussanak szükségszerűen egyszerűbbek, kompaktabbak azoknál, de a mai mobil eszközök már leköröznek a 10 éve még jónak számító asztali gépeket. Jelen írás azonban nem ezen operációs rendszerek felépítésével foglalkozik, hanem a rájuk való programfejlesztéssel. Mert míg a hardver nagyban befolyásolja, hogy mit valósíthatunk meg az eszközön, addig az operációs rendszer már a hogyanra is sokféle választ adhat.

Amikor mobil eszközökre alkalmazást írunk valószínű, hogy találkozunk majd olyan esettel, amikor alkalmazásunkat egyik eszközről a másikra kell átültetni. Ez egy sokkal triviálisabb dolog, mint személyi számítógépek között. A piac jóval töredezettebb, szükséges a jelenlét több platformon is. Ráadásul, személyi számítógépek esetében, megválaszthatjuk az eszközeinket, találhatunk esetleg olyan keretrendszereket, amellyel szinte ugyanazt a kódot fordíthatjuk több géptípusra.

Jó lenne tudni, hogy ez hasonlóképp működhet mobil eszközökön is. Ez azonban nem így

van. Eleve hatalmas a különbség egy csupán firmware-rel és egy operációs rendszerrel rendelkező gép közt a hardveres különbségektől eltekintve is, de mint látni fogjuk, léteznek köztes megoldások, amelyek akár több eszközön is működhetnek. A fejezet célja az, hogy bemutassa ezen rendszerek közti különbségeket, jelen pillanatban mik azok a trendek amelyekkel találkozhatunk, és ez mit jelent a felhasználó számára. Számunkra ez fontos tudnivaló, mert a csupán firmware-rel vagy operációs rendszerrel rendelkező gépek között jelentős különbség van.

Abban az esetben, ha nem rendelkezünk operációs rendszerrel, a programok általában egy a géptípushoz készített szoftver fejlesztői csomaggal(Software Development Kit, SDK) készülnek. Egy ilyen szoftver fejlesztői csomag számunkra legérdekesebb része az API, amelyen keresztül a hardvert manipulálhatjuk.

### 3.2 Operációs rendszerek nélküli megoldások

A mobil eszközök nagy része nem rendelkezik operációs rendszerrel. Ilyen esetben az alkalmazás egy a gyártó által a géptípushoz készített szoftver fejlesztői csomaggal(Software Development Kit, SDK) történik. Mint más platformokon, ez is biztosít számunkra egy API-t amin keresztül a géphez hozzáférhetünk. Ilyen esetekben a fejlesztés gépközeli, előfordul az assembly használata. A fejlesztő közvetlenül használhatja a memóriát, a gép vezérlése közvetlenül memóriacímeken keresztül történhet, ezen felül a programozó akár még a DMA alrendszert is vezérelheti. Különösen igaz ez a hordozható konzolokra, amiknél igyekszenek a fejlesztők a maximumot kihozni a gépekből.

Legnagyobb hátránya ennek a módszernek, hogy teljesen gép specifikus. Egy cégen belül is jelentős különbségek lehetnek az API-k közt, és egy újabb termék gyakran nem visszafele kompatibilis az előzővel. Ez extra terhet ró a programozóra, mivel egy termék megjelenéséve egy új API megtanulásával jár, illetve a kódok is csak jelentős átírás után futhatnak egy másik rendszeren.

### 3.3 Symbian OS és a Symbian Platform

A mobil operációs rendszerek piacán a legrégebbi és legnépszerűbb szereplő a Symbian. Pár éve még egy zárt operációs rendszer volt, de az utóbbi időben átalakult a Symbian platformmá, s 2010 februárja óta open source.

A Symbian platformként sok lehetőséget ad a programozó kezébe. Támogat több multiplatform megoldást. Támogatja a Flash Lite,Java ME, Qt , WRT technológiákat, rendelkezik Python és Ruby implementációkkal, emellett rendelkezik egy natív megoldással, amit Symbian C++-nak hívnak.A rendszer képes multitaskingra és ismeri a multitouch technológiát is. Ismeri a widget fogalmát, a programozók fejleszthetnek saját widgeteket. A technológia támogatja a számítási felhőket, nagyban épít a webes technológiákra. Ennek eredménye az lehet, hogy a megjelenítés könnyen cserélhető a telefonok közt, míg a lényegi technológia minden készüléken ugyan az marad.

Amennyiben a Symbian natív megoldását szeretnénk használni, 3 segédeszközt kaphatunk hozzá. Ezek az Application Development ToolKit(ADT),Software Development Kit(SDK) és a Product Development Kit(PDK). Az ADT tartalmazza mindazt, ami a Symbian-es programok létrehozásához kell. A használt IDE  a Carbite.c++ ami az Eclipse-nek egy a Symbian-re szabott verziója, GCC fordítót használ és debug segédeszközök is járnak hozzá. Az SDK tartalmazza az alkalmazásfejlesztéshez szükséges API-kat, míg a PDK a Symbian platform bővítésére szolgál és elsősorban a driver és a middleware fejlesztőket célozza meg. Az ilyen módon készült kódok nagy része nem hordozható, de a jövőben a Qt teljes részét képezi majd a rendszernek, ami jelentősen javít a szituáción. A Symbian-es alkalmazások terjesztése a Symbian Horizon-on történik, de más úgy nevezett Marketplace-ek is léteznek, mint például a Nokia Ovi-ja.

### 3.4 Windows Mobile 6.x, Windows Phone 7, Windows CE

A Microsoft azokban az időkben lépett a mobil eszközök piacára, amikor még a Palm uralta a piacot. Nem konkréten egy PDA-kra készült operációs rendszerrel, hanem egy általános beágyazott rendszerekre készült megoldással, a Windows CE-vel rukkolt elő. Fontos megemlíteni, hogy a Windows CE nem a cég egy nagygépes operációs rendszer  beágyazott megoldása, hanem egy teljesen külön fejlesztést. Mivel ez egy általános megoldás több konkrét megvalósítása létezik. Régebben például külön volt PocketPC és Smarthphone vonal, de ahogy a két kategória egybeolvadt úgy a két megvalósítást is a Windows Mobile váltotta fel, amelyet a 7-es verziótól Windows Phone 7-nek hívnak.

A Windows Mobile aktuális verziója a 6.5.3-as, de a 7-es verzió jelentős újításokat fog

tartalmazni. A 7-es verzió nem lesz kompatibilis a régebbi verziókkal. A 6.x verzió multitask-os operációs rendszer, a PDA-kból való örökségnek fő beviteli eszköznek a stylus-t tekinti. A 6.5-ös verzióval kerültek az ujjal való használatot megkönnyítő megoldások, olyanok mint az új start menü, a widgetek támogatása, teljesen újragondolt menürendszer. Azonban a fejlesztők az új verzió adta lehetőségeket nem tudják alkalmazni, a fejlesztett alkalmazások hacsak nem saját megoldást használnak a régi kinézettel jelennek meg.

Az alkalmazások fejlesztése Visual Studion keresztül támogatott, de a 2010-es verzió már nem tartalmaz lehetőséget Windows Mobile 6.x verziókra való fejlesztésre. A Windows Mobile 6.x-re fejleszteni natív kódban C++-ban lehet, de a .NET keretrendszernek is létezik egy mobil eszközökre szánt verziója a .NET CF. A Windows Phone 7-től kezdve az alkalmazás fejlesztés központjában a Silverlight és a .NET lesz. A Silverlight egy webalkalmazás keretrendszer az Adobe Flash technológiájának ellenfeleként jelent meg 2007-ben. Képes együttműködni a .NET keretrendszerrel, vagy szerver oldalon, vagy, ha az eszköz támogatja, akkor kliens oldalon is. Ez azért fontos, mert a Windows Phone 7 alkalmazások GUI-ját is ez működtetné. A .NET CF támogatás megszűnik, a .NET keretrendszer azon verziója kerül, amelyet a Microsoft másik 2 nagy platformja, az asztali operációs rendszer, és az Xbox család is használ. Ezzel együtt a .NET keretrendszerhez tartozó XNA is integrálásra kerül. Az XNA elsősorban a játékfejlesztőknek nyújt támogatást gyors és hatékony játékfejlesztésre.

A Microsoft fontosnak tartja kiemelni, hogy az eddigi funkció központú interface-ekkel szemben, a Windows Phone 7 információ központú lesz. Ez azt jelenti, hogy nem egy alkalmazást indítunk el, és azzal manipulálunk adatokat, hanem az adatok közt lapozgatva konkrét információkon manipulálhatunk. Az adatok pedig úgynevezett hubokban lesznek, amelyek közt lehetnek átfedések. Egy konkrét példával élve fényképek közt szemezgetünk, egyik képen ott van egy barátunk, és eszünkbe jut, hogy milyen rég beszéltünk vele. Ha ez a fénykép a közösségi oldalakon már ismerhető módon az adott fényképen megjelölt, a fényképről indíthatunk hívást az illető fele, vagy épp más a személyhez kötött információkhoz juthatunk. Hasonlóan egy zeneszám lejátszása közben képesek lennék kapcsolatba lépni másokkal akiknek tetszik a dal, vagy épp a zenekar fellépéseiről informálódni. Ezek mind alkalmazásként futnának, és a Symbianhez hasonlóan itt is a megvalósítás nagyon gyakran webalkalmazásokat takarna.

A Windows Phone 7 ezzel együtt kevesebb is lesz. Nem lesz Flash támogatás, ez a Silverlight miatt érthető is. A visszafele kompatibilitás hiányát már említettem. De ezen túl a programok is csupán a Windows Marketplace-en keresztül lesznek elérhetőek, és csak úgy, mint az App Store esetében, amelyről még később lesz szó, az alkalmazások egy úgy nevezett approval process-en mennek keresztül, amely során a Microsoft elbírálja, hogy a program felkerülhet-e a marketplace-re. Ez a végét jeleni a Java-nak és a Qt-nek is, melyeknek eddig létezett Windows Mobile-os implementációja. Mindezeken felül a hagyományos multitasking is kikerül a rendszerből. Bizonyos alkalmazások ugyan képesek lesznek a háttérből is működni, mint a zenelejátszás, böngésző, de a külső fejlesztések közül adott időpillanatban csak egy futhat majd. Ezzel a rendszer válaszidejét akarják csökkenteni, az alkalmazások indítását felgyorsítani.

A Windows Phone 7 még nincs kész, de a Microsoft a Zune HD-t és a Microsoft Kin platformokat bemutatásakor már kinyilvánította, hogy az ezen eszközök fejlesztése során felhalmozott tapasztalatokat használják majd fel a Windows Phone 7 készítése során. A Zune HD a média tartalmakra koncentrál, míg a Kin a szociális oldalakkal való együttműködést részesíti előnyben.

### 3.5 BlackBerry OS

A RIM saját fejlesztésű operációs rendszere a saját készülékein elérhető. Kezdetektől vállalati színtérre szánták, központban az email-ezéssel. Így a készülék tökéletesen működik együtt a Lotus Notes, Novell Groupware és Microsoft Exchange csoportmunka szoftverekkel. Az alkalmi felhasználók számára várhatóan a 6-os verzióval kerül be a Facebook, Twitter, RSS támogatás. A rendszer multitaskingos, a 6-os verziótól kezdődően pedig multi-touch kijelzővel és mozgásérzékelős görgetéssel rendelkezik majd.

A Blackberry OS 2 -féle fejlesztési hozzá állást kínál fel: egy web alkalmazás alapút és egy a platformon nagy múlttal rendelkező Java alapút. A web alkalmazások lehetnek böngészőből elérhetőek, de léteznek Blackberry-s widgetek. Ezek külsőre és viselkedésre hasonlóak a többi Blackberry-s alkalmazáshoz, de a háttérben HTML, CSS és JavaScript dolgozik. A fejlesztés a Java JDK 1.6 -ra épül, ehhez társul a Blackberry Widget SDK. A támogatott fejlesztői környezetek az Eclipse és a Visual Studio. A Widget-ek valós hardveren való telepítéséhez a RIM

jóváhagyása szükséges. A Java alapú fejlesztés MIDP 2.0 és CDLC 1.1 alapokon történik. A támogatott IDE az Eclipse. Minden BlackBerry OS 5.0-val ellátott eszköz támogatja a JSR 239-en keresztül az OpenGL ES 1.0, saját megvalósítással rendelkezik az SQLite-ra és rendelkezik speciális API elemekkel amelyek nem találhatóak meg a Java-s specifikációban. A RIM ezen megoldását RIMletnek hívja. A RIM nem gátolja a MIDlet-ek fejlesztését, azonban a MIDlet-ek a speciális biztonsági feltételeknek és a Blackberry OS sajátosságai miatt nem feltétlenül fognak az elvártnak megfelelően működni. Lehetőség van a JAR fájlokat a RIMlet-ekhez hasonlatos CODokba konvertálni. Ez az úgynevezett sign-olást, tehát biztonsági elfogadtatást lehetővé teszi, de a tökéletes felhasználó élményhez még egyéb finomhangolás szükséges. A fejlesztőnek magának kell végiggondolnia van-e szüksége a RIMlet-ek által nyújtott jelentős többletszolgáltatást, vagy inkább a multiplatform fejlesztést helyezi előtérbe.

### 3.6 iPhoneOS

Az iPhone OS-t az eddig felsorolt operációs rendszerek közül időben legkésőbb jelent meg a piacon, de megjelenése az egész piacot felforgatta. Ugyan az iPhone OS visszalépés volt a vele egy időben piacon levőkhöz képest abban, hogy nem támogat multitasking-ot, se Adobe Flash-t. Ezzel szemben az iPhone jött először elő teljesen ikonalapú interfésszel, amelyet multi-touch gesztusokkal lehet vezérelni, annak köszönhetően pedig, hogy az alkalmazások, bizonyos belső szoftverek kivételével, párhuzamos futása nem volt megengedett javított a rendszer válaszidején, és kiiktatta a sok szoftver által okozott belassulásokat. Az iPhone OS-el lett népszerű a vásártér, vagy MarketPlace, ami az iPhone OS esetében az App Store. Alkalmazás csak az App Store-on keresztül telepíthető a készülékekre.

Az iPhone OS elsősorban az iPhone okostelefonhoz készült, de később az iPod Touch-ot és az iPad-et is ez éltette, és élteti most is. Speciális követelmény a felhasználók részére, hogy mindig a legújabb verziójú operációs rendszerük legyen. És mivel a különböző hardver közt nincs lényeges eltérés, így a programozó számára nem szükséges külön optimalizálni a különböző eszközök között, ami eddig egyik oprendszerről sem volt elmondható. A programok Objective-C -ben készülnek, és az Apple az iPhone SDK-t nyújtja hozzá, a támogatott IDE az Xcode 3.1. Az iPhone OS nem támogat más megoldásokat. Nem használható rajta se Java, se

.NET, se Flash. Mivel az Apple-ös megoldások csak a cég termékein használhatóak, csak jelentős munka árán lehet a programokat portolni más nem Apple rendszerekre.

Az egész rendszer megvalósítása egyben azonban mégis sikeres, mivel a felhasználók számára egy sokkal intuitívabb interface-t kínál, és az App Store-on keresztül a fejlesztő is jobban el tud jutni a felhasználóhoz. Ráadásul az egyező hardver-OS páros miatt nem tapasztalhatók kompatibilitási problémák a rendszer és szoftver között, ami még inkább hozzájárul a jó felhasználói élményhez.

### 3.7 Maemo és a Meego

A mobil operációs rendszerek piacán hosszú ideje léteznek Linux alapú megoldások, de komoly piaci szereplése még csak az utóbbi időben kezdődött meg. A Maemo a Nokia mobil Linux megvalósítása, amely 2005-ben mint tablet OS jelent meg, és csak 2007 óta használják okostelefonokon is. Jelenlegi verziója a Maemo 5, de 2010 februárjában a Nokia az Intel-lel egy közös sajtókonferencián bejelentette, hogy egyesítik a Nokia megoldását az Intel Moblin-jával. Ennek eredménye a Meego, és a fejlesztés ezentúl ezen a vonalon halad tovább.

A Maemo első különlegessége, hogy nagyon közel áll az asztali gépekhez. Egy Debian Linux disztribúción alapszik, és elméletileg ugyan az a kód lefut az asztali gépen és az okostelefonon is. Gyakorlatban sok esetben működik is, esetleg egy x86 helyett ARM-re kell fordítani az alkalmazást, de itt is ajánlott figyelembe vennie a programozónak a kis kijelzőméretet, és az esetleg korlátozott beviteli perifériákat. A Maemo is képes multitasking-ra, de a multi-touch technológia csak az elkövetkező release-kbe kerül majd bele. A Maemo-s fejlesztésekre több fejlesztőeszköz áll rendelkezésre. A Nokia saját megoldása a Maemo SDK, amely C alapú, és kiemelten támogatja a Qt-t, de közösségi szinten a GTK+ is támogatott, a fejlesztéshez az ESBox IDE nyújt segítéset, mely Eclipse alapokon nyugszik. Java alapú alkalmazások is fejleszthetőek a Jalimo Maemo SDK segítéségével. Ez azonban nem a mobil Java megoldásra, a Mobile Edition-re, hanem a Standard Edition-re épül. Ezen kívül a Maemo támogatja a Ruby-t, a Python-t és a Mono-t is.

### 3.8 Android

Az Android a Google Linux kernelre épülő operációs rendszere. Köszönhetően a Google nevének nagy érdeklődést keltett bejelentése óta, a piacon 2008-ban jelent meg. A platform Java-s alkalmazások futtatását támogatja, de ezek nem az Oracle(Sun) által felkínált Java Mobil Edition-ön, hanem egy speciális Dalvik virtuális gépen futnak. Az Android támogatja a multitouch-ot és a multitasking-ot. A hivatalosan támogatott IDE az Eclipse, amely az Android Development Tools Plugin-en keresztül válik alkalmassá Androidos alkalmazások fejlesztésére. Az Android alkalmazások fejlesztéséhez a Google a Android SDK-t biztosítja. Bizonyos részei az alkalmazásoknak, vagy könyvtárak készülhetnek más nyelven is(elsősorban C-ben), ezeket az Android Native Development Kit segítségével fordíthatjuk ARM natív kódra. Az alkalmazások terjesztése az Android Market-en keresztül lehetséges, amelyen a Google nem gyakorol olyan jellegű ellenőrzést, mint például az Apple vagy a Microsoft a saját megoldásain. Az Android technológiát még később tüzetesebben is szemügyre vesszük írásomban.

### 3.9 Qt

A Nokia által birtokolt Qt technológia egy multiplatform megoldás a szoftverfejlesztésben, elsősorban felhasználó felületek megvalósítására. Olyan szoftverek mögött áll, mint a Google Earth, KDE, Opera vagy Skype. Nagyon fontos előnye, hogy ez egy nyílt és ingyenes technológia, amelyet több platformon elérhető, többek közt mobil eszközökön is. A Nokia mind a Symbian mint a Maemo/Meego platformján elsődleges fejlesztési eszközének tekinti, de a 6.x-es Windows Mobile verziókra is telepíthető.

### 3.10 Java

A Java Mobile Editon-t külön kerül kifejtésre az Androiddal együtt, de hogy a többi szoftverplatform kontextusába lássuk, szükséges már most megismernünk bizonyos alapvető tulajdonságait.

A Java Mobile Edition a „write once, run everywhere” szellemiségében készült, egy olyan multiplatform keretrendszer amely képes alkalmazásokat futtatni a lehető legtöbb különböző mobil eszközön. Mivel a mobil eszközök közt hatalmas különbségek vannak, ezért egyértelmű, hogy ez a feladat teljes egészében nem megvalósítható. Mégis elmondható, hogy a Java Mobile

Edition a mobil eszközök jelentős hányadán megtalálható, és az alkalmazások viszonylag jól hordozhatóak egyik rendszerről a másikra. A Java sikere annak köszönhető, hogy valóban a kisebb teljesítményű eszközökön is el fut, ezzel szemben az eddig említett megoldások csupán a felső kategóriát célozzák meg. A Java Mobile Edition alkalmazások fejlesztése Eclipse-ben és Netbeans-ben történhet, de az Oracle által elsődlegesnek tartott platform a Netbeans. A Java-s alkalmazások terjesztéséhez nincs konkrétan meghatározott lehetőség.

### 3.11 Az operációs rendszerek és a főbb technológiák összegzése

A hardvereknél már kitértem arra, hogy a mobil technológiák terén egyszerre figyelhető meg egyes platformok zártabbá válása, amivel párhuzamosan régi és új nyílt platformok jelennek meg. A gyártók még keresik a sikerrecepteket, a piac nagyon képlékeny, a technológiák gyorsan változnak. Nekünk, programozók számára pedig az lenne a legjobb, ha minél több platformon lenne elérhető az általunk készített program. Az előbbiekből látható, hogy a piac teljes lefedése lehetetlen, a legnagyobb fejlesztőcégek sem tudják azt biztosítani. Platformok közt még nyelvi szinten se tudunk mozogni, hisz nagyon gyakran a platform által támogatott programozási nyelv is más. Ezáltal a programozóknak több nyelvvel is meg kell ismerkednie, hogy az egyes platformokra fejleszthessen, és természetesen a platform API-jával is tisztában kell lennie. A fejlesztő környezetekről általánosan elmondható, hogy a hordozható konzolok kivételével ingyenesek. Az azonban már jóval gyakoribb, hogy a programok vásártérre való feltöltéséért fizetni kelljen. A zárt rendszereknek elfogadott, hogy a piactér használatáért, ami ráadásul az egyetlen módja a programok terjesztésének, a hardvergyártó részesedést kap. Ezért cserébe a fejlesztő biztos terjesztési csatornát kap, és a hardvergyártó próbál lépéseket tenni a kalózkodás megakadályozására. A nyílt rendszerek nagy előnye, hogy a fejlesztőrendszerek is nagyon gyakran nyíltak, és így a fejlesztőrendszerek fejlesztése a közösség által támogatott. Emellett sok kód megjelenik open-source, vagy épp a készítő kínálja fel annak lehetőségét további felhasználásra ingyen.

Maguk az egyes platformokon használható API-k maguk is több API-ból épülnek fel. Egy API-n belül lehet külön Security API vagy Bluetooth API, és az már csak az adott platform szabványától függ, hogy ezek közül melyeket kötelező támogatnia a készülékeknek, s melyek

opciónálisok. Amennyiben opcionális API-t használunk, az természesen leszűkíti az elérhető készülékek számát. Bizonyos platformok esetén pedig ezen API-k megvalósítása sem egyforma minden rendszeren, ez alatt pedig főképp a Java érthető, ahol a virtuális gépek közti eltéréseknek köszönhetően a program nem várt dolgokat produkálhat. Ezáltal újra ki kell hangsúlyozni mobil eszközökön a nem-emulátoros-tesztelés fontosságát.

A mobil eszközökre való fejlesztés 2 új iránya a Wi-Fi hálózatok terjedésén, a 3G technológián, a mobil böngészők fejlődésén és a számítási felhőkön alapszik. Ez a két új irány a mobil eszközökön is megtalálható widgetek megjelenése, másrészt pedig a böngészőn keresztül vett webszolgáltatások jelenléte. Böngészős változatban kérdéses lehet, hogy az adott mobil eszköz böngészője mennyire támogatja a technológiák. Erre jó példa, hogy az Apple már most sem támogatja, a Microsoft pedig a jövőben nem fogja támogatni a Flash technológiát. A widgetek létrehozására, már ahol van rá lehetőség, saját fejlesztőkörnyezet támogat. Így a widgeteket ugyan minden készülékre el kell készíteni a munka mégis jóval kevesebb, mint ha a teljes alkalmazást kellene újraimplementálni.

# 4. Java

## 4.1 Java mobil eszközökön

Az előző fejezetben taglaltak szerint jelenleg 4 -féle Java implementációval találkozhatunk mobil eszközökön. Ebből a Linux-os platformok a Java Standard Edition-t használják, amely a desktop-os Java környezetet jelenti. Az Oracle(Sun) hivatalosan mobil eszközökre való megoldása a Java Mobile Edition, erre épít rá a RIM saját megoldásával. A Java nyelvet használja fel, de sem a Standard Edition-re, sem a Mobile Edition-re nem épít az Android megoldása. E négy technológia közül a Java Mobile Edition-re és az Android kerül tárgyalásra az elkövetkezőekben.

## 4.2 A Java Mobile Edition felépítése

A Mobile Edition mobil eszközökön 2 külön specifikációban valósul meg. Ezek egyike a CDC, vagyis a Connected Device Configuration, míg a másik a CLDC, ami a Connected Limited Device Configuration-t takarja. A CDC elméletileg nagyobb teljesítményű mobil eszközökre készült, közelebb áll az asztali platformhoz, míg a CLDC a limitált képességű mobil

eszközöket célozza meg. Feltehetőleg a CLDC nagy népszerűsége miatt azonban jelenleg a nagyobb teljesítményű okostelefonokon is főleg a CLDC implementációi vannak jelen. A CDC fő erőssége, hogy a Personal Profile API-n keresztül a Standard Edition-ös AWT-vel találkozhatunk. Ennek egy leszűkítése a limitáltabb képességekkel rendelkező eszközök számára a Personal Basis Profile. Ez egyrészt a programozó számára is jó, mivel az API ismerős lesz, másrészt a felhasználónak sem lesz teljesen új a GUI. Hátránya az, hogy ez nem egy mobil eszközökre tervezett felület, amely miatt lehetséges, hogy nem is a legintuitívabb megoldásról van szó. A Personal Basis Profile-on és a Foundation Profile-on alapszik az Advanced Graphics and User Interface. Ez tekinthető a legfejlettebb Mobile Edition-ös felület programozó API-nak. Ennek segítségével már Swing-es osztályokat is használhatunk, és a programunk felülete valóban az asztalos verzióhoz fog közelíteni külalakban.

A CDC azonban más megvalósítást követ. A CDC-re épülő API a MIDP. A MIDP elkészítésekor központi szerepet kapott, hogy a készülékek széles skáláján fusson. A MIDP minden egyes verzióhoz létezik egy minimumkövetelmény, amely a kor mobil eszközeinek egy optimális minimumát próbálja szimbolizálni. A MIDP 1-es és 2-es verziója így akár a 2 színű megjelenítéssel rendelkező 96x54 -es felbontású eszközökön is elérhető. Az 1.0-ás verzió ezen felül még nem támogatta a lebegőpontos számokat, az active rendering-et, nem lehetett lekérdezni bármikor a billentyűk állapotát. A 2.0-ás API számos újítást hozott, itt már támogatott a HTTPS és a GUI-n is fejlesztetések történtek. Ami a legjelentősebb, hogy az API részét képezi két új elem, a multimédia API és a Game API. Ezek speciális multimédiás funkciók megvalósításánál igazán használhatóak. Fontos megjegyezni, hogy a MIDP és a CLDC bizonyos mértékig szabadon kombinálható, és készülék specifikus, hogy milyen megvalósításban lesz elérhető. Konkrét példaként, egy készülék mely CLDC 1.0 és MIDP 2.0 szabványokat támogat, ugyan hozzáfér a MIDP 2.0 által nyújtott előnyökhöz, de a lebegőpontos számításokhoz CLDC 1.1 verzió szükséges.

Mindezeken felül a CLDC felépítése nagyban moduláris, egyéb API-k is léteznek, amelyek nem tartoznak a MIDP magjába. Ilyenek például a SVG API, amelynek segítségével képesek vagyunk SVG grafikát megjeleníteni alkalmazásunkban, vagy az M3G, amely 3 dimenziós grafikák is megjeleníthetőek, de léteznek API-k amelyeket a multimédiás vagy a biztonsági

szolgáltatásokat bővítik ki, illetve kihasználják a mobiltelefonos Bluetooth-ban rejlő lehetőségeket. A készülékek ezek közül nem mindet valósítják meg, ez függ a készülék képességeitől, és a megvalósítástól. A Sun mellett gyártó specifikus API-k is léteznek, ezekkel általában a gép specifikus tulajdonságait használhatjuk ki.

### 4.3 A MIDlet, felépítése és életciklusa

A MIDP alkalmazásokat MIDlet-eknek hívjuk. Nem csupán nevükben hasonlatosak az appletekhez, de szerkezetükben is. Minden MIDlet fő osztálya a MIDlet osztályt terjeszti ki. Az alkalmazás futásának kezdetén ez példányosul. A MIDlet osztálynak 3 absztrakt metódusát kell implementálnunk. Ezek a startApp(), pauseApp() és a destroyApp(). Alkalmazásunk futása a startApp() funkcióval kezdődik. Ennek futásával a MIDletünk active státuszba kerül. Az alkalmazásunk ezen kívül még paused vagy destroyed státuszban lehet. Minden egyes alkalommal amikor az alkalmazás háttérbe kerül átkerül paused státuszba, és lefut a pauseApp metódus, visszatérve aktívba pedig újra meghívódik a startApp. Ezáltal lehetséges, hogy a háttérben levő alkalmazás ne használjon felesleges erőforrásokat, megállítsuk az interakciót igénylő folyamatokat míg a felhasználó visszatér vagy a rendszer által elvett erőforrásokat visszatéréskor visszaszerezzük. A destroyAppal van lehetőségünk az alkalmazásunk szabályos befejezését kiváltani. Fontos megjegyezni, hogy a háttérben futó alkalmazások eldobódhatnak, ha a rendszernek extra erőforrások bevonására van szükség. Ebben az esetben is ez a függvény hívódik meg.

### 4.4 GUI programozás MIDP-ben

A MIDP 2 -féle megközelítést kínál a GUI programozására. Egy alacsony és egy magas szintű elképzelésről van szó, a két megközelítés ötvözése egy alkalmazáson belül lehetséges, azzal a megkötéssel, hogy a két megjelenítési mód elemeit nem lehet vegyíteni. A magas szintű megvalósítás direkt mobil eszközökre optimalizált GUI komponenseket tartalmaz, míg az alacsony szintűnél konkrét lehetőségünk van a kijelzőre rajzolni. Hogy rajzolhassunk a képernyőre a Display osztálynak kell megszereznünk egy objektumát. A Display osztály a fizikai kijelző reprezentációja, és minden egyes MIDlet-hez példányosodik egy belőle. A Display-en a Displayable osztály leszármazottjai helyezhetőek el, mint megjelenítendő felületek. A Displayable 2 leszármazottja a Screen és a Canvas. A Screen tekinthető a magas szintű API

ősosztályának, míg az alacsony szintű megjelenítésre a Canvas osztály leszármazottjait vagy az abból származtatott saját osztályokat használhatjuk. A Screen leszármazottjai alapvető képernyőelemek. Ezek a Form, List, Alert és a TextBox. Ezen osztályok közül a Form egy olyan osztály amely különböző megjeleníthető elemeket tartalmaz. Ezek azok a GUI komponensek, amelyekkel a legtöbb GUI-ban találkozhatunk. A megjeleníthető GUI komponenseknek az Item osztály az ősosztálya. Az ebből származtatott osztályok a ChoiceGroup, DateField, Gauge, ImageItem, TextItem, StringItem, TextField. Ezek segítségével megvalósíthatjuk a legalapvetőbb GUI elemeket a kijelzőn. Ha ez azonban mégsem lenne elég, a CustomItem osztály származtatásával és a draw() metódusának implementálásával saját elemeket hozhatunk létre. Az egyes komponens osztályok példányait az append() függvény segítségével csatolhatjuk egy formhoz. Hogy ezek a komponensek hogy fognak megjelenni, a Form automatikus elrendezése fogja eldönteni. Az elhelyezkedést befolyásolhatjuk az egyes komponenseknek azok setLayout() metódusán keresztül. Az ezen keresztül megadott javaslatokat a tartalmazó objektum figyelembe. Az alacsony szintű Canvas osztály származtatott osztályaihoz nem csatolhatjuk ezen elemeket. A Canvas elemeinek kialakítását teljesen nekünk kell implementálnunk. Cserébe hozzáférhetünk a teljes képernyős módhoz, a közvetlen újrarajzoláshoz és ha a készülék képes rá double buffering alkalmazásához. A Canvas paint() metódusának implementálásával saját magunk valósíthatjuk meg a megjelenő képet.

Az input kezelése interfészeken keresztül történik. Az egyes komponensek, és maguk a Screen és Canvas osztályok képesek eseményeket dobni a rajtuk bekövetkezett interakciónak köszönhetően. Ezen üzenetet a megfelelő interfész implementálásával tudjuk lekezelni, majd az implementáló osztálynak fel kell iratkoznia az egyes komponensnél az üzenetek fogadására. A legegyszerűbb kommunkáció a Command osztályon keresztül történik. Lehetőségünk van Command példányokat adni Displayable objektumoknak, de Item-eknek is. Egy ilyen Command objektum létrehozásakor 3 paramétert adhatunk meg, egy stringet, amely a Command címkéje lesz, egy típust és egy prioritást. A típus és a prioritás határozza meg, hogy a rendszer mely gombokra osztja ki ezeket az inputokat, a kijelzőn hogy jelennek meg, de a Command azonosításában is segítenek. A típusok példaként megemlíteném a Command.EXIT, Command.ITEM típusokat. Egy Command.EXIT típusú Command megjelenítésben a készülék

hagyományosan Exit gombját használhatjuk majd nagy valószínűséggel, és kijelzőn is valószínűleg a megszokott helyen találhatjuk majd. Az ITEM speciális szerepe pedig az, hogy ezzel az Item-eken végrehajtott interakciókat jelzi a rendszer, és például egy Item kijelölésekor jelenik meg. Az egyes Item-ek azonban önmaguk is küldenek eseményeket, amikor rajtuk interakció történik, ezek kezelése pedig az ItemStateListener interfész implementálása segítségével lehetséges.

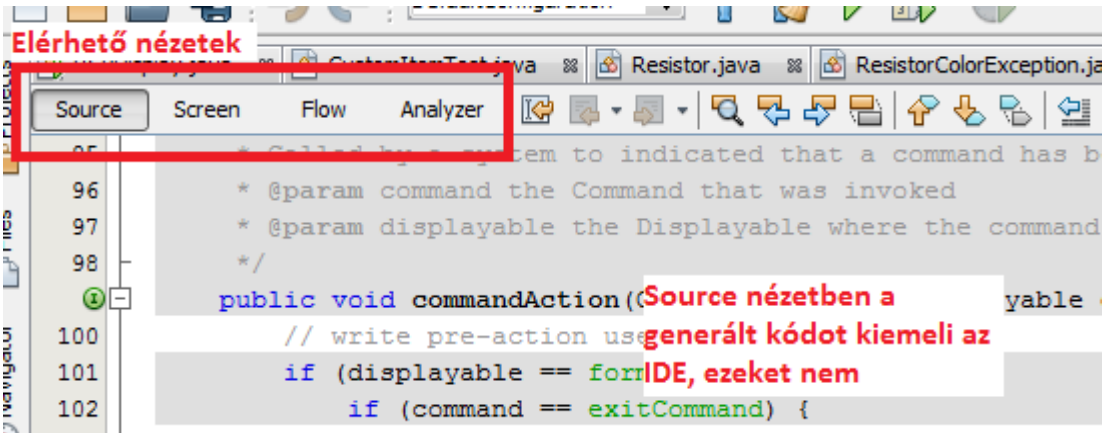
A Canvas osztály, mint Displayable gyermeke, szintén képes a Command-ok fogadására, teljes képernyős módban azonban a felhasználó számára a Command-ok láthatatlanok. A Canvas osztály azonban rendelkezik olyan metódusokkal, amelyekkel a készülékből érkező eseményeket lehet kezelni. Ezen metódusok felülírásával képesek vagyunk a saját eseménykezelőnk megírására. Emellett a Canvas osztály lehetőséget biztosít bizonyos game action-ök kezelésére. A game action-ök egy absztrakciót képeznek a gép és a Java közt, így a programozónak nem a tényleges billentyűzetkiosztással, és a gombok megléte miatt kell aggódnia, hanem az implementációra bízhatja ezek kezelését. A Canvas egy származtatása a GameCanvas, amely a MIDP 2.0 óta van jelen. A MIDP 2.0 megjelenésével a Sun a mobil játékfejlesztés irányában jelentősen kibővítette a MIDP képességeit. A Game API a 2 dimenziós játékok fejlesztését hívatott elősegíteni, és olyan osztályok tartoznak bele, mint a GameCanvas, a Layer, Sprite, LayerManager és a Layer osztály egy kiterjesztése, a TiledLayer. A GameCanvas jellegzetessége a még tovább egyszerűsített input handling és double buffering.A Canvas nagy hasznunkra válik, amennyiben különböző grafikákat akarunk megjeleníteni. Azonban minél inkább alacsonyszintű megoldásokat választunk, annál inkább a programozóra hárul a készülékek közti különbségek kezelése.

### 4.5 A Java Mobile Edition és a NetBeans

A Java Mobile Edition által elsődlegesen támogatott fejlesztői környezet a NetBeans. A Netbeans hivatalos oldalán eleve találunk a Java ME-vel is felszerelt csomagot, de ha lenne már a gépünkön egy telepített NetBeans Java ME nélkül, akkor egyszerűen a pluginok közt megkeresve automatikusan települ.

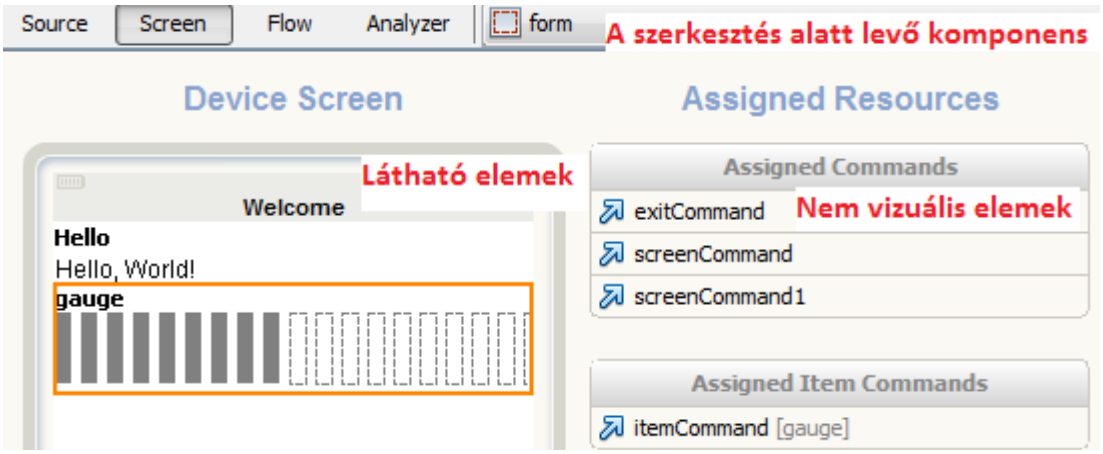
A NetBeans a Java ME-s alkalmazás fejlesztés megkönnyítésére tartalmaz egy Visual Mobile Designer nevezetű eszközt és egy GameBuilder nevezetű eszközt. Az elsővel a

hagyományos alkalmazások, míg a második a játékok fejlesztését támogatja. A Visual Mobile Designer 4 képernyővel rendelkezik a MIDlet szerkesztésére. Az első nézetet „Source”-nak hívják, és mint neve is mutatja, ebben láthatjuk programunk forráskódját. Ez a nézet úgy van kialakítva, hogy a „Screen” és „Flow” nézetekben összeállított programnak megfelelő struktúrát a NetBeans generálja. A generált kódon a szövegszerkesztőből nem tudunk változtatni, kivéve ha az egész Visual Designer-t kikapcsoljuk, a viselkedés definiálása azonban kódszinten történik.




*1. A Source nézet*

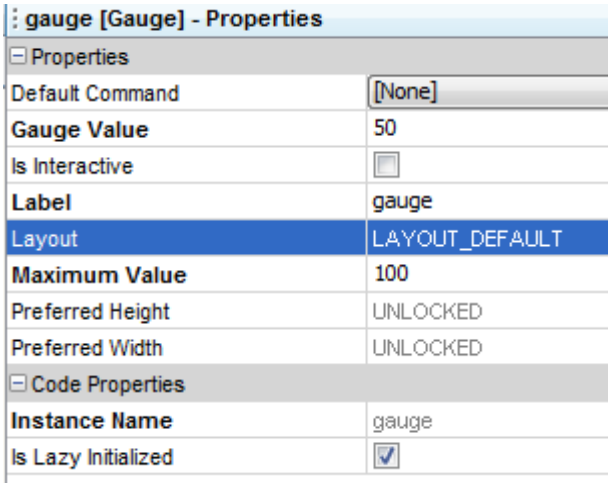
A „Screen” nézeten váltogathatunk a Screen ősosztályú objektumok közt. A Screen osztályból származtatott osztályokról már fentebb megjegyeztük, hogy olyan osztályok, amelyek



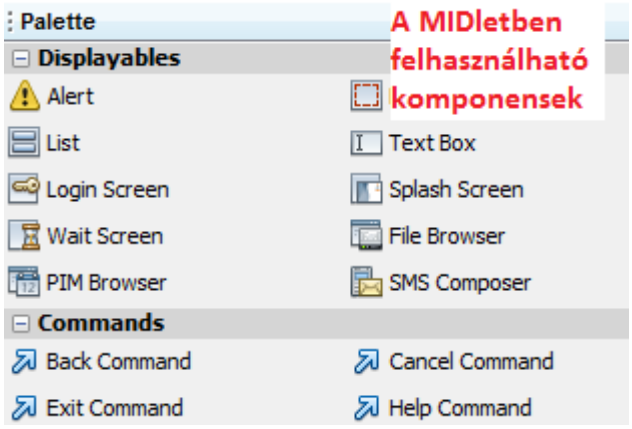

*2. A Screen nézet*

a képernyőn megjeleníthető elemeket tartalmaznak. A Device Screen alatt láthatjuk a különböző

komponensek várható elrendezését. Ebből nem tudunk vonatkoztatni az egyes komponensek tényleges külalakjára, mert az implementációnként változó. Az elemek elhelyezkedését a az alapbeállítás szerint jobb oldalt elhelyezkedő Properties ablakban tudjuk beállítani. A Device Screen mellett, az Assigned Resources alatt találhatóak az olyan nem látható elemek, melyek jelentősen befolyásolják a program működését. Ezek a csatolt parancsokat foglalják magukba, illetve a külsőleg használt erőforrásokat, mint amelyek a hangok, képek, szövegek.




*3. Egy Gauge objektum property-jei*

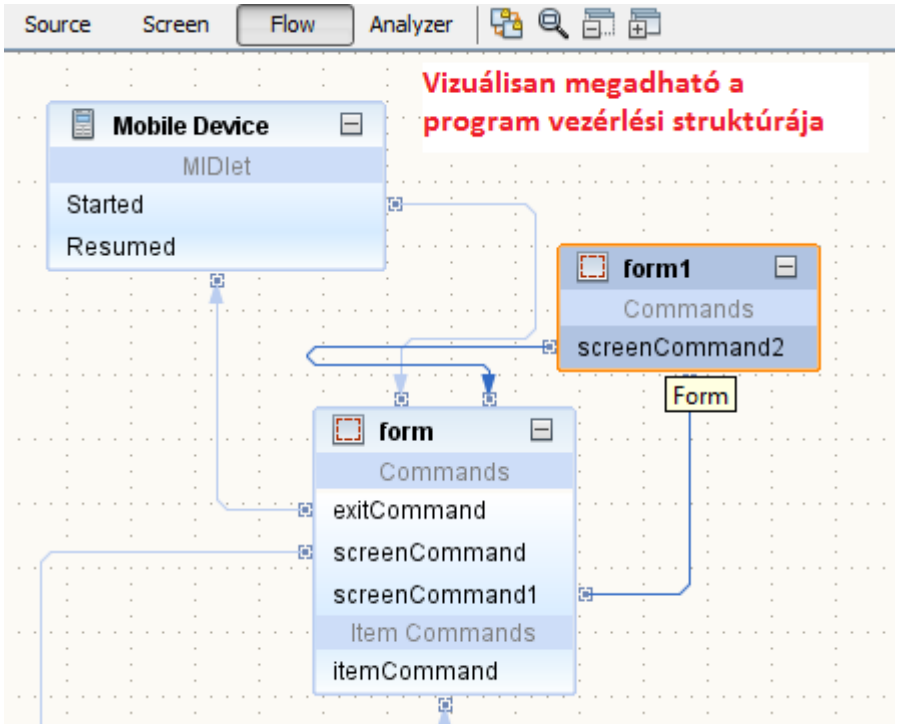



*4. Pár paletta elem*

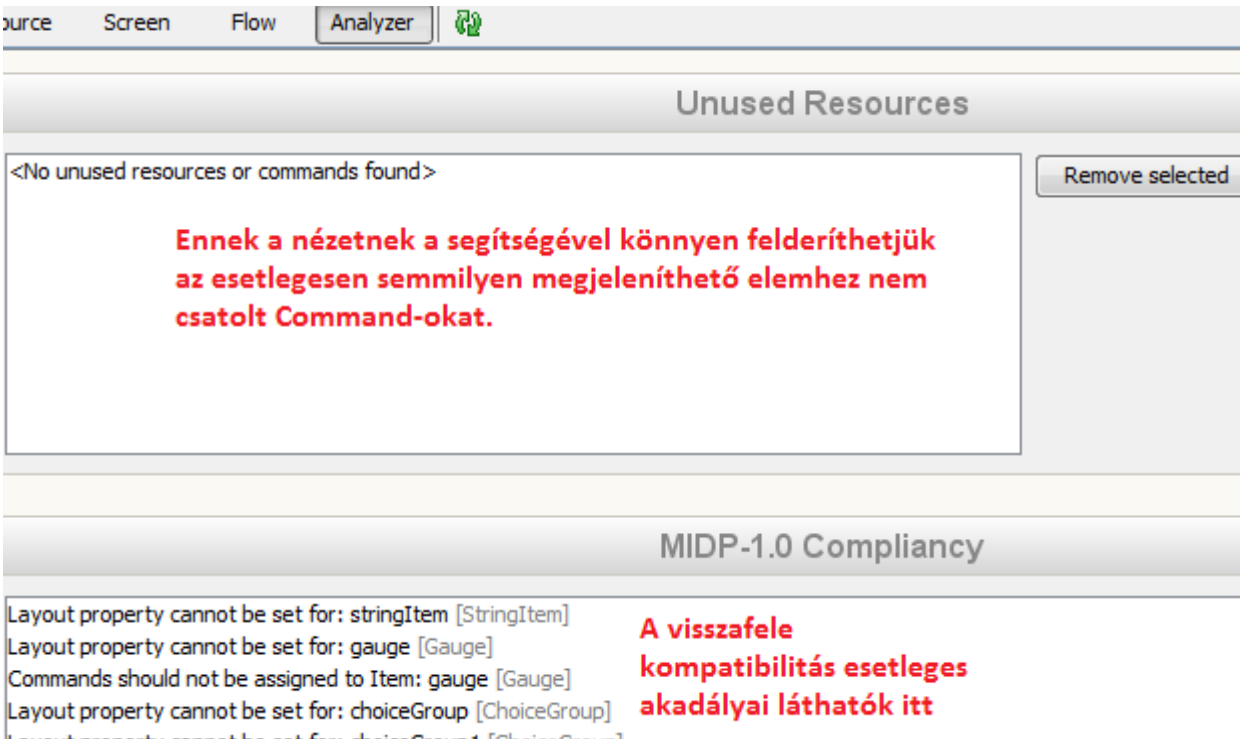
A harmadik nézet, a „Flow”, a program vezérlési struktúrájának kialakítására van. Itt nem láthatjuk az egyes Screen ősosztályú objektumok látható elemeit, viszont a hozzá csatolt Command-ok megjelenítésre kerülnek. Megjelenítésre kerülnek viszont vezérlő objektumok, melyek mögött a metódus realizálódik a kódban, mellyel egyéb feladatokat valósíthatunk meg.

Ezen a felületen egyszerű összehuzogatással kitudjuk alakítani a programon belüli navigációt, az egyéb elvégzendő feladatokat pedig a kódba írva tehetjük meg. A Visual Designer segítségével szükségszerűen összetett navigáció leképezhető. Lehetséges alkalmazásrészletet létrehozni, olyan Command-ot ami visszatérít az előző megjelenített komponenshez. Két komponens közé beilleszthetőek köztes metódusok. Ezeknek a metódusoknak a segítségével megoldható a több irányú elágaztatás.

Az utolsó nézet az Analyzer nézet, melyen keresztül a programunk ellenőrzését végezhetjük el. Ez a nézet két szempontból fontos. Egyrészt láthatjuk azon fel nem használt erőforrásokat vagy Command-okat, amelyek valamilyen oknál fogva nem csatoltak semmilyen Displayable-höz. Ennek köszönhetően a program futása során ezen erőforrásokat nem tudnánk elérni, de ezek mégis betöltődnek és értékes memóriát foglalnak. Ezeket az erőforrásokat itt egy gombnyomással eltávolíthatjuk. A másik érdekessége a nézetnek, hogy a MIDP 1.0-val való visszafele kompatibilitást is ellenőrizhetjük itt. Az itt kiírt problémákat eltávolítva és betartva az egyéb MIPD 1.0-ás szabályokat, mint például az egész aritmetikát, egy MIDP 1.0-ás platformon is működő programot kapunk.

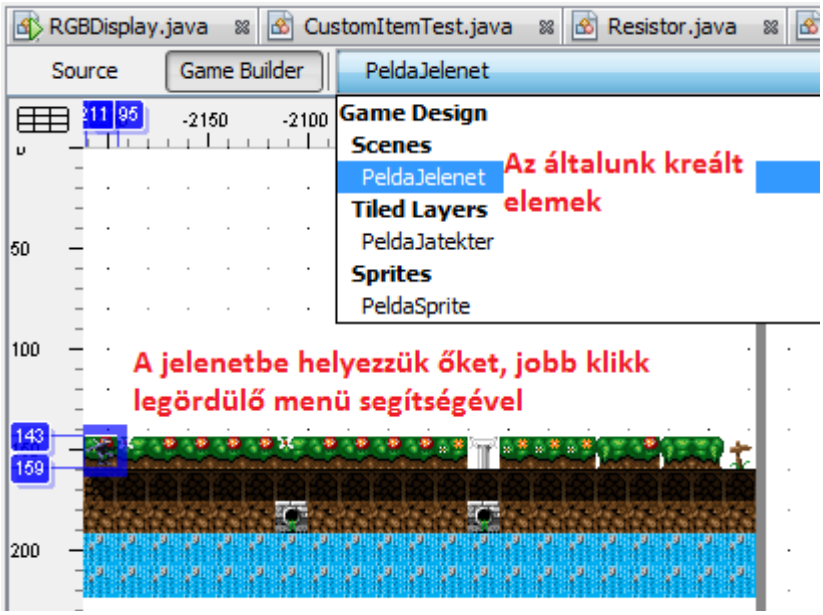



*5. Vizuális folyamattervező*




*6. Kód elemző*

A másik fontos segítségünk a mobil alkalmazás fejlesztésben a GameBuilder. A GameBuilder egy olyan eszköz, amellyel egy játék számára egy resource filet, vagy ahogy a GameBuilder nevezi, gamedesign-t kreálhatunk. A GameBuilder 2 dimenziós játékok elkészítését támogatja, 3 különböző elemet ismer. Az első a Scene, avagy jelenet, amely az egy LayerManager objektumhoz tartozó elemeket foglalja magában. A



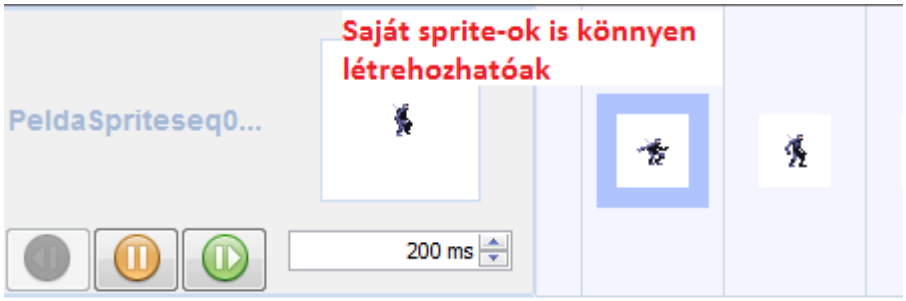

*7. Jelenet szerkesztő*

LayerManager egy olyan osztály, amely a Layer osztály objektumait foglalja magába. A Layer-ök a képernyőn rétegenként elhelyezkedő megjeleníthető elemek, melyek egy adott sorrendben rajzolódnak ki, így az előrébb levő rétegek elfedik az alsóbbakat. Ez egy hagyományos megoldás a 2 dimenziós játékoknál.

A második elem a Tile Layer-öket foglalja magában. A TiledLayer osztály egy objektumát állíthatjuk elő ennek segítségével. A TiledLayer egy térképként képzelhetjük el, mely kis képkockákból épül fel úgy, hogy a különböző képkockákat letároljuk egy helyen, majd a térképen ezekre a képkockákra már csak hivatkozunk. Így egy képben tárolhatjuk a térképen felhasznált grafikát, amely egyszerűsíti a felhasználást, gyorsítja a különböző térképek készítését és jelentős mennyiségű memóriát spórolunk meg vele.GameBuilder segít a használt kép kiválasztásában, és beállíthatjuk rajta a kívánt rácsméretet, mindezt úgy, hogy azonnal láthatjuk a várható végeredményt. Ezután egy négyzethálós felületen lehet összeállítani a kívánt térképet. Egy LayerManager-ben több TiledLayer használható. Apró trükkök segítségével így akár 2 dimenzióban is perspektíva szimulálható. Ugyanis ha a hátrább levő Layer lassabban megy mint az előtte levő, azt az ember szem távolibbnak érzékeli.

A harmadik eszköz, amit játék készítésénél felhasználhatunk, a Sprite editor. A Sprite editor segítségével apró mozgó figurákat hozhatunk létre képkockákból, amely a játékos karaktert, és ellenfeleit szimbolizálhatja. A Sprite editor az elkészített Sprite-okból a Sprite osztály egy példányát hozza létre. A Sprite-oknak elkészíthetjük az animációs fázisait, kiválaszthatjuk hozzá a képet, láthatjuk magát az animációt is. A Sprite animációjának késleltetése public integerként van letárolva, ugyanis a Sprite animációt nem valósítja meg automatikusan a Game API, a képek léptetéséről a programozónak kell gondoskodnia.

*8. Tile editor*




*9 Sprite editor*

Már készítés alatt is megtekinthető a látvány mögé generált kód. A szerkesztő által

```
public Sprite getPeldaSprite() throws java.io.IOException {
    if (PeldaSprite == null) {
        // write pre-init user code here
        PeldaSprite = new Sprite(getPlatform_tiles(), 16, 16);
        PeldaSprite.setFrameSequence(PeldaSpriteseq001);
        // write post-init user code here
    }
    return PeldaSprite;
}
```

egy sprite kialakítása

*10. És a mögöttes kód*

létrehozott kódrészletek itt sem módosíthatóak, de a legtöbb cselekvés elé és után saját kódot illeszthetünk. A fentebb látható getPeldaSprite metódust a mégfentebb látható PeldaSprite szerkesztéséből kaptuk. A getPlatform_tiles() metódus az általunk kiválasztott kép erőforrást szedi össze, szintén automatikusan implementálódik, a Sprite kiterjedésére vonatkozó paramétereket pedig a Sprite létrehozásakor kell megadni. A framesequence-hez egy integer tömb generálódik, ez adódik át a setFrameSequence metódusnak. A frame delay pedig osztályszintű globális változóként tárolódik a kód elején. Az IO exception dobása abban az esetben következik be, ha a getPLatform_tiles() nem tudja lefoglalni a képet, mint erőforrást.

A jelenet szerepe a különböző elemek összerakásánál van szerepe. Nem szükséges ugyanis a jelenet különböző elemeit egyenként beolvasni. A jelenet létrehozásával generálunk egy metódust ami ezt megteszi helyettünk. Fontos, hogy a dobott IO kivételek elhagyják a GameDesign-unkat, azt képesek vagyunk magasabb szinten kezelni. A LayerManager pedig

paraméterként jelenik meg. Ennek következtében egy IO kivétel esetén képesek vagyunk a játékost visszairányítani a menübe, vagy egy másik jelenet betöltésébe foghatunk. A programunk írásánál tehát a dolgunk az, hogy példányosítsuk a GameDesign-unk, majd a LayerManager-ünkre hívjuk meg ezt a metódust. Így a jelenetünk felépül. A játéklogikát pedig erre a jelenetre elérjük. Az olyan információkat, amelyekre a játék logikájának megvalósításában szerepe lehet pedig elérhetjük a megfelelő metódusok hívásának, és a publikus osztálytagoknak köszönhetően.

A jelenet összeállítása

```
public void updateLayerManagerForPeldaJelenet(LayerManager lm) th
    // write pre-update user code here
    getPeldaSprite().setPosition(-2211, 143);
    getPeldaSprite().setVisible(true);
    lm.append(getPeldaSprite());
    getPeldaJatekter().setPosition(-2212, 0);
    getPeldaJatekter().setVisible(true);
    lm.append(getPeldaJatekter());
    // write post-update user code here
}
```

*11. A jelenet felépül, elemenként van letárolva*

### 4.6 A Resistance Calculator program, a program tervezése, UML diagramja

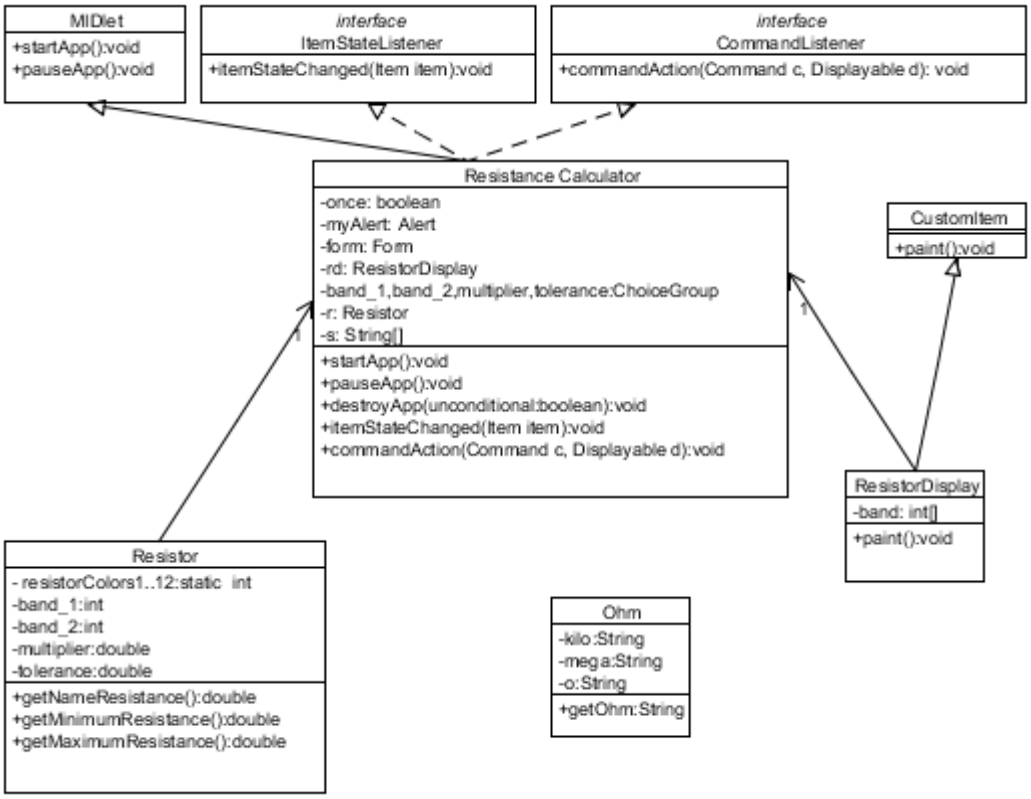
A fenti editorok sok segítséget jelenthetnek alkalmazásunk elkészítésében, de elkerülhetetlen, hogy időnként a programozónak hozzá ne kelljen nyúlni mélyebben a programhoz, vagy az alkalmazás egy részét segédeszközök nélkül kell összehoznia. Mivel a MIDP tervezésében fő kívánalmat jelentett az egyszerűség ez sem okoz nagy nehézséget. Az elkövetkezőekben egy MIDP-s alkalmazás fejlesztésének folyamatát mutatnám be egy példaprogramon keresztül. A program ötlete onnan jött egy barkácsolni szerető rokonomtól jött, akinél egyszer egy áramkört raktunk össze, és az ellenállások értékeit egy excel táblázatból néztük ki. Ez akkor nem jelentett nagy gondot, de nagy könnyebbséget jelent, ha magunknál is tarthatunk egy ilyen alkalmazást.

A Resistance Calculator alkalmazással szembeni követelmény az volt, hogy képes legyen 4 sávos ellenállásokat azonosítani színkód alapján, úgy, hogy mondja meg az ellenállásról a

nevesített ellenállását, és a tűrést, amiben belül a valós érték mozoghat.

A követelményekből kiindulva elkezdtem megrajzolni a GUI-t papíron. Tervem szerint a színeket egy legördülőmenüből lehet kiválasztani sávonként, és a kiválasztott színeket visszajelzéskép külön meg is jelenítem. Ezen kívül a felhasználó külön utasításként adja majd ki az Ohm érték kiszámítását. Ezekből a gondolatokból kezdtem el kialakítani az UML diagrammot. A program belépési pontja a MIDlet osztály egy származtatott osztálya. Azon belül is a startApp() metódus. Ezt az osztályt ResistanceCalculator-nak neveztem el a program után. Úgy terveztem, hogy az eseményvezérlést is ezen az oszályon keresztül oldom majd meg. Mivel a GUI-elemek kezeléséhez Commandok-ra és és Item-ekre lesz majd szükségem, így az ItemStateListener és CommandListener interfészek implementálására lesz majd szükség. A megjelenítés és a logika kettéválasztása érdekében egy Resistor osztályt és egy ResistorDisplay osztályt hoztam létre. A Resistor osztály az ellenállás éppen aktuális állapotát tartalmazza, míg a ResistorDisplay osztály segítségével az aktuális állapotot megjeleníthetem. Ezen kívül azt szerettem volna, ha mértékegység szerint 2 tizedes pontossággal tudtam volna megadni az eredményt. Ennek érdekében egy Ohm nevű konvertáló osztályt hoztam létre, mely a getOhm() metódusával a megadott double paramétert Stringgé alakítja.

Ismerve az API lehetőségeit, úgy gondoltam a számomra legmegfelelőbb elem egy legördülő lista számára a ChoiceGroup osztály lenne, így azt is hozzávettem az UML-hez. Míg a ResistorDisplay osztály a CustomItem osztályból származtatva lett, mivel csak ezzel tudok saját grafikát létrehozni a Form-okon. Az egész hátterét, mint már pár oldallal feljebb kiderülhetett egy Form szolgáltatja majd, hisz ez az az elem, amelyre a MIDP GUI komponenseket rá lehet akasztani. Ezen kívül az eredmény megjelenítését bíztam egy külön Alert-re, hogy ne egy képernyőn zsúfolódjon össze minden.

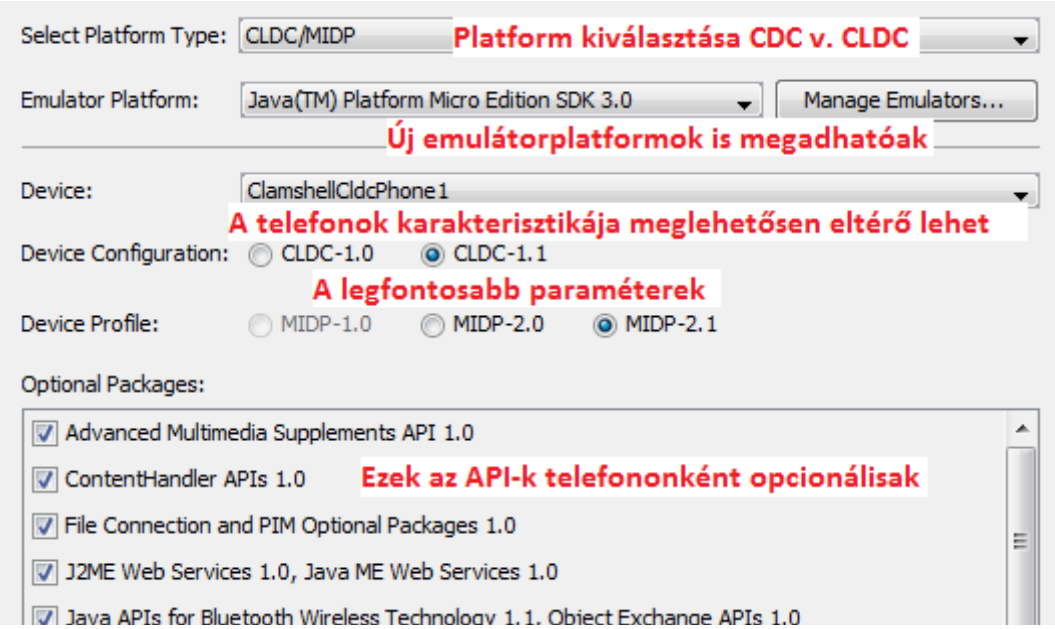



*12. A Resistance Calculator UML diagramja*

### 4.7 A program megvalósítása

Miután kész a terv, az implementálás előtt már csak egy lépés van hátra, a megfelelő projekt beállítások kiválasztása. Ez a NetBeans Projekt létrehozásakor lehetséges, de később is módosíthatunk rajta a Project->Properties alatt. A beállításokról lentebb látható egy screenshot. az egyes beállítások elnevezése az eddigiek alapján egyértelmű lehet, így csupán a beállítások döntésére térnék ki. Az alapötlet az volt, hogy CLDC/MIDP telefonokra készül az alkalmazás, így ezen beállítás nem volt kérdéses. A lebegőpontos számítás szükségessége miatt minimum CLDC 1.1 konfigurációra van szükség. A MIDP 2.1 nem indokolt, de mivel a készülékek legjava már támogatja, ezért ezt az opciót az alapbeállításon hagytam. Az emulátor beállítások lényegtelenek jelen program esetében, mivel a magas szintű API következtében minden

telefonemulátoron használható programhoz kell jutnunk. Ezt a végén megvizsgáljuk majd. Még opcionális API elemeket választhatunk ki, de ezekre jelen programban nincs szükség, így azok tetszés szerint kikapcsolhatók. Megjegyzendő azonban, hogy minden beállításunk a kiválasztott emulátor függvényében módosul.




*13. Változatos beállítási lehetőségek*

A kód kialakítása az UML-nek megfelelő. Az alkalmazásunk központi osztálya a MIDlet, implementálja az UML-en megjelelölt interfészeket. A startApp() függvény tartalmazza az erőforrás inicializálásunkat. Az egyszer inicializált erőforrások kerülhetnek ide, a konstruktorba

```
public class ResistanceCalculator extends MIDlet
        implements ItemStateListener, CommandListener
```

*14. Az input feldolgozását a MIDletünk végzi*

vagy egy inicializáló blokkba is. A GUI összeállítása során először példányosítjuk a különböző elemeket, és a megfelelő állapotba állítjuk ezeket, ha szükséges. Ilyen például a legördülő menük, amelyeket értékekkel kell feltölteni. Szerencsés esetben ezek iteratívan megoldhatók.

```
public void startApp() {
    if (!once) {
        once = true;
        Command c = new Command("Exit", Command.EXIT, 0);
        Display.getDisplay(this).setCurrent(form);
        rd = new ResistorDisplay("Resistor display");
        band_1 = new ChoiceGroup("Band 1", List.POPUP);
        band_2 = new ChoiceGroup("Band 2", List.POPUP);
        multiplier = new ChoiceGroup("Multiplier", List.POPUP);
        tolerance = new ChoiceGroup("Multiplier", List.POPUP);
        r = new Resistor();
        for (int i = 0; i < 10; i++) {
            band_1.append(s[i], null);
            band_2.append(s[i], null);
        }
        for (int i = 0; i < 12; i++) {
            multiplier.append(s[i], null);
```

az alkalmazás belépési pontja

egyszer inicializálandó erőforrások

a formunk lesz látható

a menüelemek értékeit ha lehet iteratívan inicializáljuk

*15. A MIDlet inicializálása*

A vezérlés megoldása a form-okon történik majd. A fenti képen látható egy Command példányosítása, ez a form addCommand() metódusával lesz felvéve, a MIDletünk pedig a form addCommandListener() metódusán keresztül iratkozik fel az eseményekre. Az események kezelése az interfészek implementálásával történik. Lejjebb látható erre két példa. A kiváltott esemény azonosítására többféle lehetőségünk is adódik. Vizsgálhatjuk az események egyes tulajdonságait, tehát lekérdezhetjük attribútumait, vagy közvetlenül vizsgálhatjuk mely

```
ublic void commandAction(Command c, Displayable d) {
    if (c.getCommandType() == Command.EXIT) {
        this.notifyDestroyed();
    }
    if (c.getCommandType() == Command.SCREEN) {

ublic void itemStateChanged(Item item) {
    try {
        if (item == band_1) {
            r.setBand_1(band_1.getSelectedIndex());
            rd.setColor(band_1.getString(band_1.getSelectedIndex()),
```

vizsgálhatjuk az esemény tulajdonságát

vagy közvetlenül a példányt vizsgáljuk

*16. Esemény vezérlés*

referenciával dolgozunk. Ezáltal több eseményre is reagálhatunk megegyezően, de specifikusan egyes eseményekre vonatkozó viselkedést is megadhatunk.

Az UML-en jelöltük, hogy lesz egy osztályunk, amellyel saját GUI elemet valósítunk meg. A CustomItem osztályon keresztül valósíthatjuk ezt meg. A felülírható metódusok közt van olyan amely szabályozza a minimum és a javasolt méretét az Item-ünknek. Ennek fontos szerepe van a készülékek közti különbségek áthidalására. Ami a legfontosabb a paint() függvény implementálása. Ez a függvény felelős az Item-ünk közvetlen megjelenéséért. A paraméterei közt egy rajzfelülethez tartozó Graphics objektum van, amelynek segítségével a felületre rajzolhatunk, illetve 2 az Item méreteit megadó változó. A felület egyszerű, nem támogatja automatikusan a double buffering-et, magát a kirajzolást is csupán statikusan hajtja végre, a frissítést kényszeríteni kell. Ezt az ősosztály protected repaint() metódusán keresztül lehet kérni, ennek külső elérését egy refresh() függvényen keresztül tettem lehetővé, amely nem tesz mást, csupán meghívja a repaint() metódust. A Graphics osztály által kezünkbe bocsátott lehetőségek széleskörűek, ez az általános rajzoló osztálya a MIDP-nek, ugyan ezt van lehetőségünk alkalmazni Canvas felületen való rajzoláskor is. Magas szintű osztály, mellyel nem csak grafikai primitíveket, de akár bitmap-eket és szöveget is tudunk kiírni.

```
protected void paint(Graphics g, int w, int h) {
    g.setColor(band[0]);
    g.fillRect(0, 0, w / 4, h);
    g.setColor(band[1]);
    g.fillRect(w/4, 0, w/2, h);
    g.setColor(band[2]);
    g.fillRect(w/2,0,w/2+w/4,h);
    g.setColor(band[3]);
    g.fillRect(w/2+w/4, 0, w, h);


}
```

*17. Saját GUI elem rajzoló rutinja*

A megjelenítést és a tényleges logikát szétválasztva a Resistor osztályunk felelős egy ellenállás megvalósításáért. A Resistor osztály megvalósításában semmilyen MIDP-s kötődést nem található, így ez akár még egy asztali gépnél is felhasználható lenne. Az egyedüli probléma,a ami itt már nagyon érezhető volt, az, hogy a MIDP még Java 1.5-öt megelőző szabványt támogat. Ennek következtében nem használhatóak például a generikusok és az enum osztályok sem. Egy

enum létrehozása sok segítséget nyújtott volna például az ellenállások sávjainak színkódjának, elnevezésének és jelentésének összekötésére. Így a menük String elemei a MIDlet-ben lettek letárolva, a színkódok a megjelenítésnél nevesített konstansokként pedig a Resistor osztály foglalja magába a megfelelő értékeket, mint ahogy azt az alábbi ábra is mutatja.

```
public static final int RESISTOR_BLACK = 0;
public static final int RESISTOR_BROWN = 1;
public static final int RESISTOR_RED = 2;
public static final int RESISTOR_ORANGE = 3;
public static final int RESISTOR_YELLOW = 4;
public static final int RESISTOR_GREEN = 5;
public static final int RESISTOR_BLUE = 6;
public static final int RESISTOR_VIOLET = 7;
public static final int RESISTOR_GRAY = 8;
public static final int RESISTOR_WHITE = 9;
public static final int RESISTOR_GOLD = 10;
public static final int RESISTOR_SILVER = 11;
public static final int RESISTOR_NONE = 12;
private int band 1;
```

ellenállás színek megadása

*18. Enum hiányában*

Ez azt is eredményezi, hogy nem egy helyen ki kell fejtenünk, viszonylag egyszerű részeket is, így több kódsort, és nehezebben olvasható kódot kapva. A setTolerance() metódusban például színeknek megfelelően fejtjük ki a megfelelő tűrésértékeket. Minél bonyolultabb programmal állunk szemben, annál fontosabb, hogy az ilyen lehetőségeket elkerüljük, hisz tisztán látható, hogy ez az osztályok számának és méretének növekedésével a kód minőségének rovására megy.

```
switch (tolerance) {
    case RESISTOR_BROWN:
        this.tolerance = 0.01;
        break;
    case RESISTOR_RED:
        this.tolerance = 0.02;
```

Szükségszerű a hardcode?
Néha elkerülhetetlen.

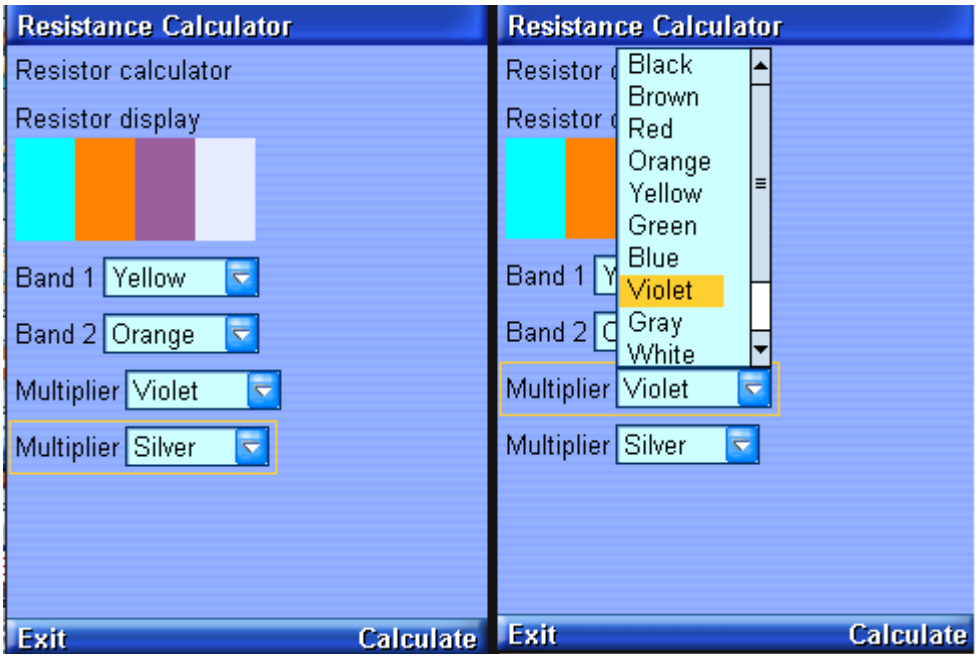
*19. Egyszerű, de nem túl elegáns megoldás*

Utolsó osztályunk az Ohm osztály lenne, amelynek szerepe csupán, hogy a megfelelő formában írja ki a kapott eredményt. Itt csupán egyszerű aritmetikai műveletekkel van dolgunk.

A kész alkalmazást kipróbálhatjuk az emulátor segítségével. A NetBeans alap esetben több

emulátort kínál fel, de ezek csupán gombjaik számában térnek el, az alkalmazás megjelenése nem fog különbözni. A valós készülékeken azonban ez már nem várható el. A különbség azonban nem csupán megjelenítésbeli, hanem funkcionális is. Például egy gauge értékét nem tudtam változtatni, mivel annak irányító gombjai nem jelentek meg a kijelzőn, és nem volt a készüléken olyan gomb, amely sliderként működhetett volna. Ez még inkább áll a a CustomItem-ekre, ahol a méretüket magunk is befolyásolhatjuk. Nem tudhatjuk, hogy fog reagálni egy készüléken a környezet, ha a kijelzőméretnél nagyobb felületű elemet hozunk létre.

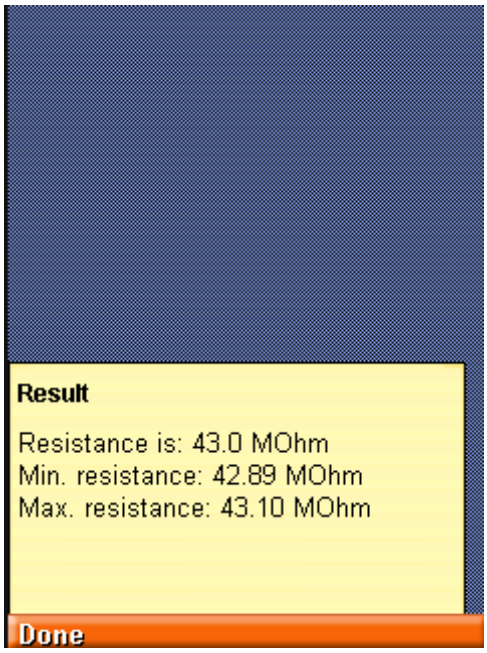
Összesítve tehát, a MIDP nyújtotta elemekkel, mindenféle külső editor nélkül is szép átlátható felületű alkalmazásokat lehet létrehozni. Az alkalmazás összeállítása a letisztult egyszerű API-nak köszönhetően könnyű. Ha tudjuk, hogy mely osztályok azok, amelyek majd az egyes eseményeket kezelik, akkor ezek objektumainak feliratkozása egyszerű metódushívásokkal




*20. A menük megjelenítése*

megoldható. A háttérben dolgozó kód nem tér el az asztali Java-tól. Az Ohm vagy Resistor osztály megoldása asztalon is ugyan úgy szükség lett volna. Továbbmenve, maga az alkalmazás design sem különbözne jelentősen egy asztali Java-s implementációtól. A legszembetűnőbb különbség, hogy nem alkalmazhatók olyan magas szintű eszközök, mint a fentebb említett

generikusok és az enum-ok. Mivel a régebbi eszközök jóval kisebb teljesítményűek voltak, így többek közt ezek is olyan overhead-et jelentettek volna, amit a MIDP kialakításakor igyekeztek elkerülni. A MIDP 3.0-ás szabvány azonban már a Java 1.6-os verziónak megfelelően fog működni.



*21. Az eredmény megjelenítése*

### 4.8 Az Android felépítése

Az előzőekben megismerkedtünk a Java Mobile Edition-nel, láthattuk nagy vonalakban, hogy milyen technológiák állnak rendelkezésünkre és elkészítettünk egy programot a rendelkezésre álló eszközökkel. Most pedig egy olyan dolog következik, amivel nagyon gyakran találkozunk a mobil alkalmazásfejlesztés során, ez pedig egy alkalmazás portolása egyik rendszerből a másikba. A másik rendszer, itt az Android-os környezet lenne. Mind a Java Mobile Edition, mind az Android Java alapú, de mégis lényeges eltérések vannak a kettő között. Fontos megjegyezni, hogy az Android egy 2007-es keltezésű rendszer, míg a MIDP 2.0-ás verziója 2002-ben született meg és 2002-ben merőben más elvárások voltak a mobil eszközökkel kapcsolatban, mint az évtized vége felé.

Az már az előző fejezetekben kiderült, hogy az Android egy Linux kernelre épül, és ezen

felül található a Dalvik virtuális gép, mely a Java-s alkalmazások futtatásáért felelős. Pontosabban minden egyes alkalmazáshoz indul egy ilyen virtuális gép, és az operációs rendszer szintjén is külön szálon indul meg a futása. Az Android sajátossága azonban, hogy az alkalmazások más alkalmazások elemeit is felhasználhatják. Ez eddig természetszerű lehetne, azonban ez nem a másik alkalmazás kódjának tartalmazásával, sem egy linkelt library felhasználásával nem jár, hanem a másik alkalmazás elindításával. Azonban ekkor nem az egész alkalmazás indul, hanem csupán annak megfelelő része. Ez pedig annak köszönhető, hogy az Android-os alkalmazások több belépési ponttal rendelkeznek. Ezeket a belépési pontokat az Android komponenseknek hívja. 4 ilyen komponenstípust különböztetünk meg.

A legalapvetőbb komponensnek az Activity tekinthető. Ezek segítségével valósíthatóak meg a grafikus interfészek, látható felületeket reprezentálnak. Ezzel ellentétes dolgot képviselnek a Service-ek, amelyek háttérben futó feladatokként léteznek. Komponensek még a Broadcast receiverek és a Content providerek. Előbbiek üzenetszórt események fogadására valók, mint például a rendszer által kiküldött alacsony akkufeszültség üzenet, vagy hogy a beépített kamerával egy új kép készült, és ezekre való reagálást oldják meg egy Service vagy Activity indításával. A Content provider pedig más alkalmazások számára teszi elérhetővé a tartalmazó alkalmazás információit, melyek valamilyen formában a háttértárolón elérhetőek.

A Content provide-rek mindig egy úgy nevezett Content Resolver kérésére aktiválódnak, a maradék három komponens azonban asszinkron üzeneteken keresztül, úgynevezett intenteken keresztül jönnek létre. Az intent-ek az Intent osztály objektumaiként reprezentálódnak, és a meghívandó komponensről tartalmaznak információkat. Ezek általában a Context osztály valamely komponens indító metódusainak valamelyikének adódnak át, mint amilyen a startActivity(),startService(),sendBroadCast(). Arról, hogy milyen komponensek elérhetőek egy adott programban, a manifest.xml fájl ad tájékoztatást. A manifest.xml az alkalmazásunk lelkének is tekinthető, a rendszer számára a futtatáshoz tartalmaz információkat. Ez tartalmazza a Java csomag nevet, ami az egész alkalmazást magába foglalja, a belépési pontokat, jogosultságokat szabályoz, de a minimális követelményként használt Android verzióról is találunk itt információkat. Itt definiálandó egy fő belépési pont is az alkalmazás számára, amely hagyományos indításkor aktivizálódik.

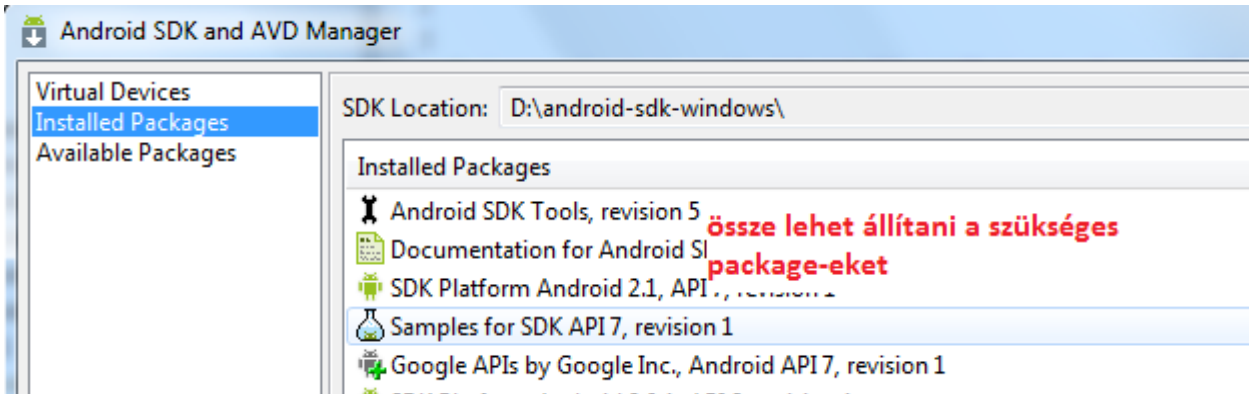
Ha egy ilyen indításkor egy Activity elindul, egy task is létrejön mellé. Az alkalmazás activity-jei általában egy task stack-et hoznak létre, amelynek során az egyazon alkalmazáshoz tartozó activity-k egy veremben tárolódnak, és csak a legfelsőn tudunk manipulálni, az az aktív activity. Ha megnyitunk egy másik alkalmazást, akkor a task háttérbe kerül, és az új alkalmazáshoz egy új task épül föl. Ha visszatérünk az előző alkalmazáshoz, akkor pedig az előző alkalmazás task-jához tartozó verem tetejéről előkerül a megfelelő Activity. Ezen kívül egy alkalmazáson belül is lehetőségünk van a multithreading-re, ennek megvalósítása szerencsére a standard Java megoldásokhoz hasonló.

Az egész komponens alapú alkalmazás szemléletnek, és az Activity-khez tartozó stack-eknek köszönhetően az alkalmazás életciklus is jelentősen más Android alatt, mint a Java Mobile Editionben. Egyrészt az életciklus különböző komponensekként. A Broadcast receiver-ek és a Service-k életciklusa egyszerű, a Broadcast reveiver feldolgozza a kapott üzenetet majd inaktívvá válik, míg a Service-k esetében a feladatuk automatikusan ismétlődhet, vagy egy a szolgáltatást hívó féltől  függően alakul. Az Activity-k szituációja attól függ, hogy épp aktívak-e, láthatóak a képernyőn vagy háttérben vannak. Ha a felhasználó épp az adott Activity-vel kommunikál, az adott Activity aktívnak számít. Ha elveszti a fókuszt, de nem teljesen fedi el a másik Activity, akkor az Activity szünetel, míg ha teljesen láthatatlan lesz a képernyőn, akkor megáll. Ezekből az állapotokból visszatérhetnek az Activity-k, még akkor is ha a rendszernek memóriára van szüksége. Ekkor az Activity állapota lementődik, majd ha újraindítjuk az Activity-t, tárolóról újra betöldődik.

Az egész komponens mizéria egyik legfontosabb eleme, hogy szétválasszuk egymástól a funkciókat, és közben több alkalmazás számára tegyük elérhetővé azokat. Hasonló modularitásra való törekvés figyelhető meg a GUI komponenseknél is. A programozói és a grafikusi szerepkör szétválasztásának fontos lépése, hogy a felületeket nem kódból, hanem XML-ből definiáljuk. A grafikus összedobálhatja a felületet egy külön szerkesztő segítségével, és az XML nyelv élsajátításával mindent kedvére szabhat. A programozó pedig ezen elemeket egyszerűen lekérheti az XML fájlból, pontosabban az abból generált resource fájlból, és a logika implementálásnál szükségtelen tudnia, mi hogy fog megjelenni. Hogy ez hogyan valósul meg, a ResistorCalculator alkalmazás portolásánál pontosan meglátjuk.

### 4.9 Az Android és az Eclipse

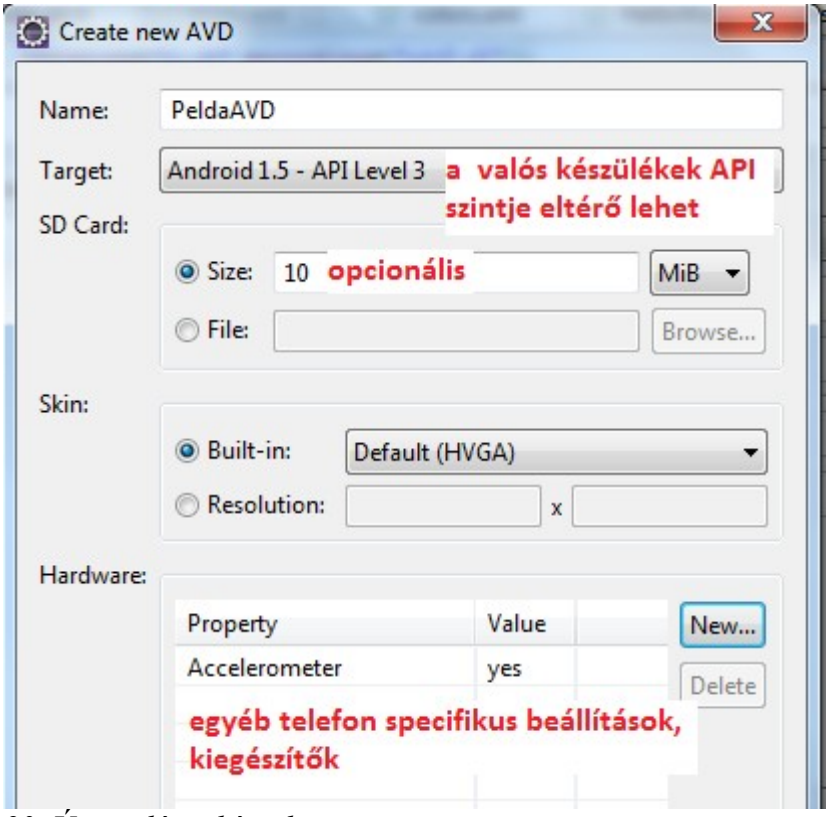
Az Android hivatalos fejlesztőkörnyezete az Eclipse, amelybe az Android pluginként települ. A plugin-on kívül szükség lesz az SDK-ra, ennek helyét kell beállítani az Eclipsben. Az




*22. Az Android API-k konfigurálása*

érdekesebb rész ez után következik. A plugin részét képezi ugyanis az Android SDK and AVD Manager, amelyen keresztül konfigurálhatjuk részletesebben az Androidot. Ezen keresztül letölthetünk adott számú API verziókat, dokumentációkat példákat.

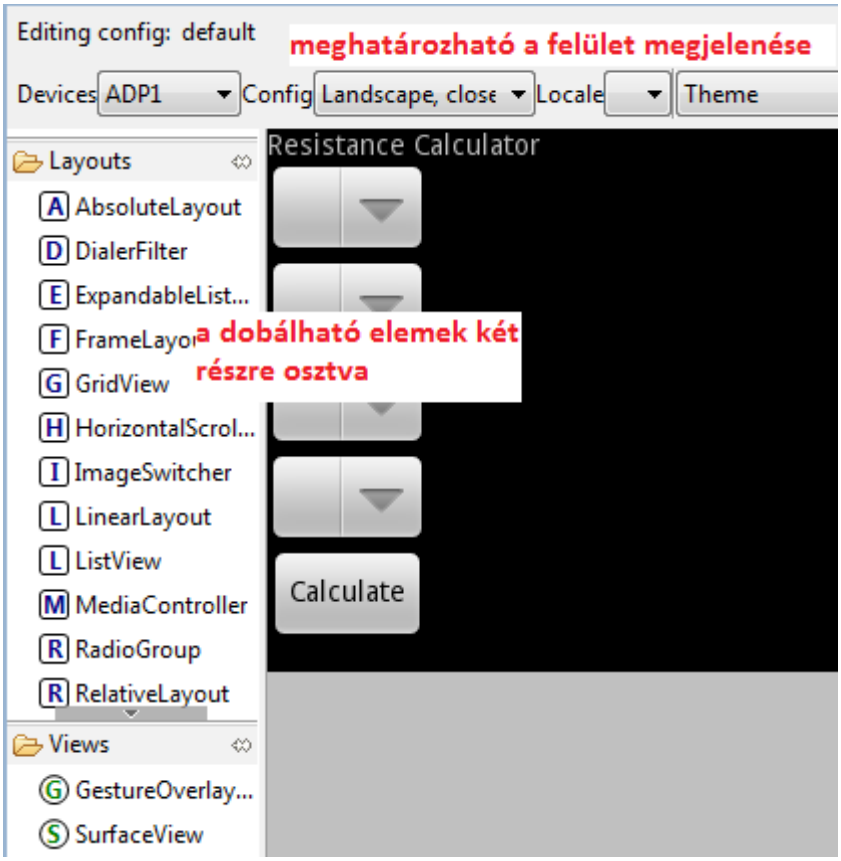
Ha letöltöttünk valamilyen API verziót, akkor létre kell hoznunk az eszközt, amelyen alkalmazásunk futni fog. Ennél fontos figyelembe vennünk az API verziót, amin az alkalmazás futni fog, mivel a készülékek nem feltétlen a legújabb változattal vannak felszerelve. A képernyő és egyéb kiegészítők hozzáadására csak speciális alkalmazás fejlesztésénél van




*23. Új emulátor készül*

szükség.Jellemző módon az Android emulátor felállása a Java Mobile Edition-ös emulátorok indulási idejének többszörösét veszi igénybe, ráadásul a gépigénye is jóval magasabb. Az emulátor először elindítja a rendszert, majd utána telepíti a készülékre az alkalmazást. Épp ezért javasolt inkább csak az alkalmazást leállítani, mintsem minden futásnál az egész rendszert újraindítani. Mindenesetre ez jóval hamarabb visszaveszi az elhamarkodott fordításokat, mint a Mobile Edition esetében.

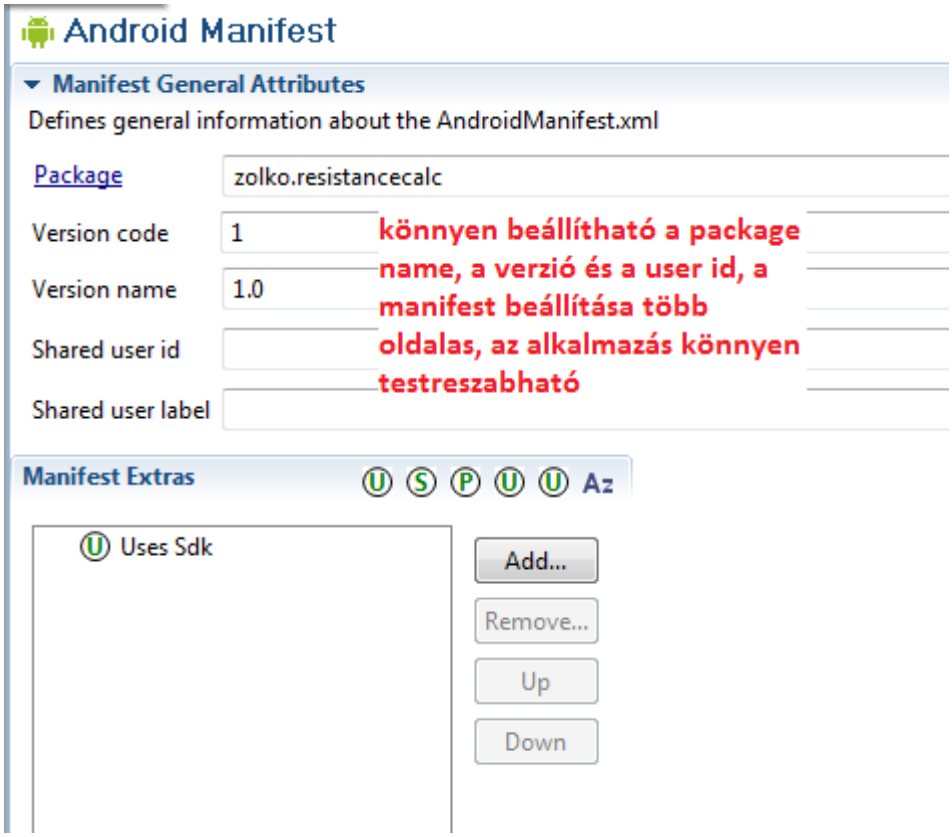
Az Androidnak nincs kifejezetten játékokra kihegyezett része épp ezért az IDE sem tartalmaz semmi ilyen eszközt, azonban adott egy grafikus felület az alkalmazások összedobására. Ezen keresztül meghatározható az egyes Activity-k kinézete, a képen nem látható, ámde jobboldalt található Properties fülön pedig a felhelyezett elemek tulajdonságait szabhatjuk testre.




*24. Layout editor*

Az alkalmazás által használt XML-ek fölé kényelmes szerkesztőfelületet ad az Eclipse, a

fentebb látható GUI tervező mögött is egy XML fájl áll, amely bármikor megtekinthető és szerkeszthető. Többek közt a szöveges erőforrásokat, vagy a manifest fájl szerkesztését is nagyon kényelmes editorok teszik lehetővé. Apróság, de gyakorlatban mégis zavaró tud lenni, hogy ezen nézetekből nem fordítható a program, a fordítás hibát jelez, így észben kell tartani, hogy a lassú fordítási folyamat előtt leváltsunk az XML fájlokról.



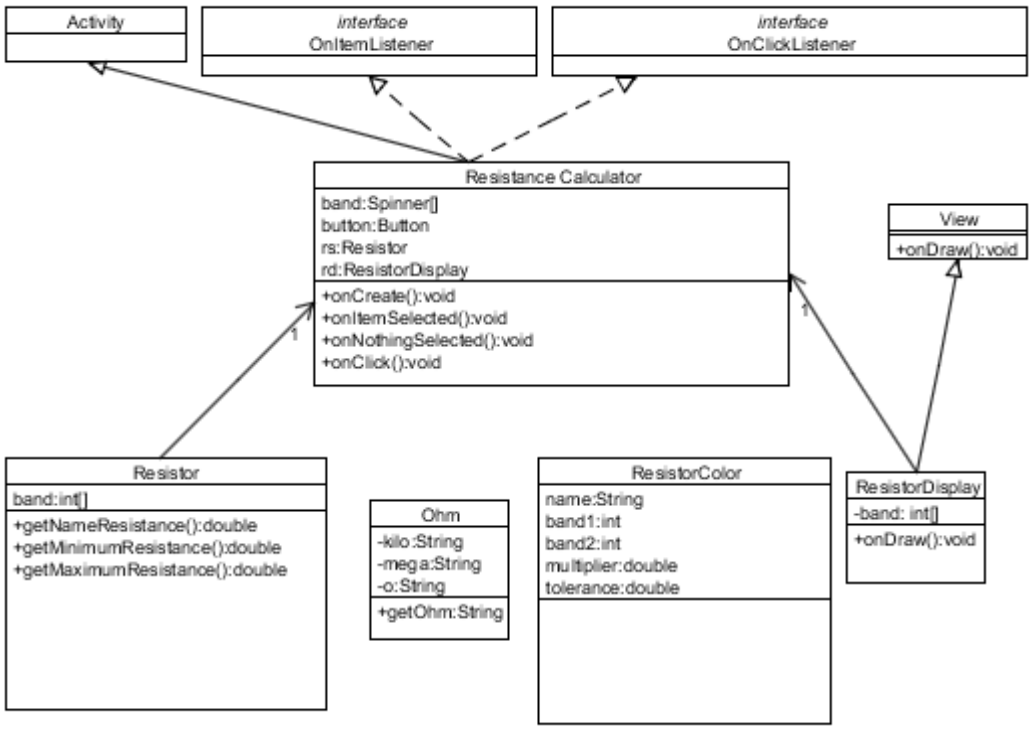

*25. Manifest editor*

### 4.10 A Resistance Calculator portolása

Az eddigiekkel sikerült megszereznünk azt a minimális információt, amivel egy Android-os alkalmazás létrehozásakor foglalkoznunk kell, de az API szerkezetéről, a tényleges kódolásról még nem esett szó. A szemléletesség kedvéért a portolással párhuzamosan mutatom be ezeket az elemeket.

Az alkalmazás szerkezete nem változik látványosan, legfőképp egyes osztályok cserélődnek

ki úgymond Android-os megfelelőjükre. Mivel egy megjelenített alkalmazás lesz a Resistance Calculator, ezért egy Activity komponens fogunk létrehozni számára, az inputot pedig továbbra is a fő osztály tartalmazza, így a megfelelő interfészeket is ez az osztály implementálja. Az Android is felkínál hasonló menüelemeket, mint a MIDP, ezért azokat is egyszerűen kicseréljük, még a testre szabott elemünk ezúttal nem CustomItem lesz, hanem egy View. Apróbb optimalizáció a kódon, mely a teljesítményt nem befolyásolja, hogy nem különálló elemekként kezeljük a sávokat, hanem egy tömbben tároljuk azokat. Ez az előző megvalósítás során szerzett tapasztalatokból fakad, és nem kötődik az Android jellegzetességeihez. Egy új osztály kerül azonban implementálásra, ami egy enum osztály. Kihasználva a magasabb Java verziót, amit az Android használ, így alkalmazható az az elképzelés, amit már az eredeti alkalmazásnál is szerettem volna megvalósítani.




*26. Az Androidra módosított UML diagram*

A fentebb már látott manifest fájl editoron keresztül beállíthatjuk az alkalmazásunk tulajdonságunk, vagy közvetlenül is manipulálhatjuk az xml-t. A manifest nagyon fontos eleme a

package, ez a sor azonosítja a rendszer számára az alkalmazást. Mindig 2 részből kell hogy álljon az alant láthatóknak megfelelően. Az alkalmazás leírása ezután kezdődik, a komponensek felsorolásával. Az alkalmazás nevét itt állíthatjuk be, az 'android:label' sor a lefoglalt erőforrás nevét jelöli, így jelenik meg az erőforrás fájlban is. Majd a komponenshez tartozó intentek következnek, ahol részleteződik milyen eseményekre indul a komponens.

Ha a programunk konfigurálásával megvagyunk, előállítjuk a szükséges erőforrásokat. Itt jelen

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="zolko.resistancecalc"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
        <activity android:name=".ResistanceCalculator"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCH
            </intent-filter>
        </activity>
```

alkalmazás csomag, verzió

alkalmazás neve

a fő intentünk

setben a

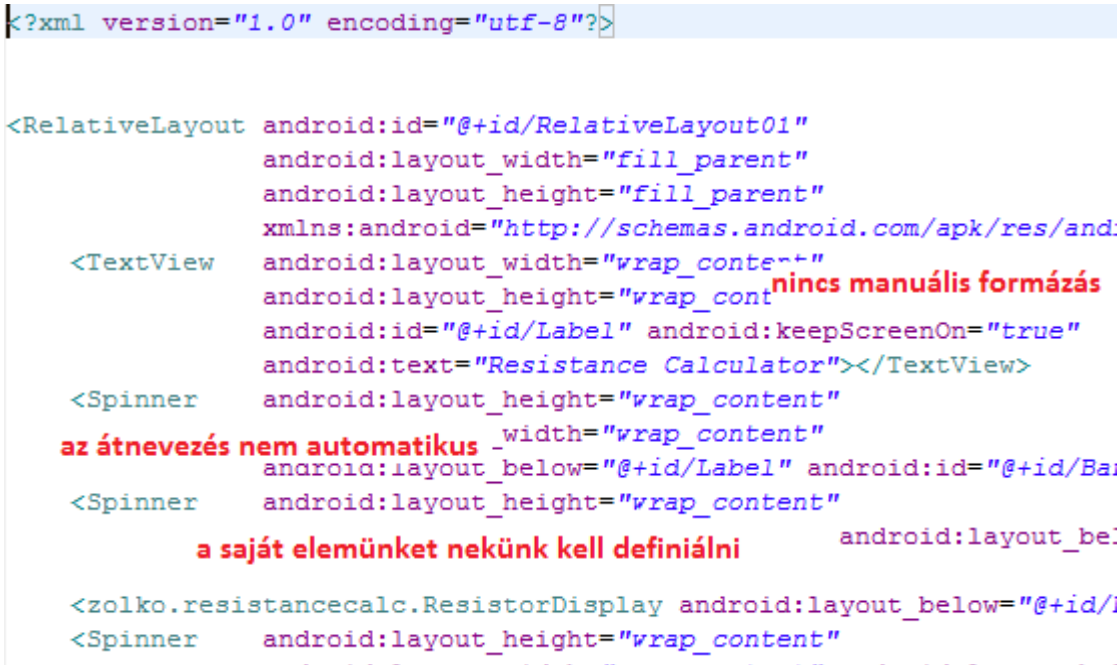
*27. A manifest fájl kinézete*

megjelenítendő felület tekinthető a legnagyobb erőforrásnak, amelynek létrehozása egy layout-on


*28. A tervezett felület*

keresztül fog történni. Fenbtebb láthatóak voltak a layout elemek, amelyeket drag and drop

módszerrel ráhúzogálunk a felszínre. A jobb oldali Properties panelen beállíthatjuk az elemek tulajdonságait, felüldefiniálhatjuk az elemek elhelyezkedését. Az összerakott elemekből a rendszer xml-t generál. Amelyet bármikor megtekinthetünk és szerkeszthetünk. Ajánlott odafigyelni, hogy az IDE nem nyújt ehhez a részhez olyan szolgáltatásokat mint a refractoring avagy a kódformázás. Ha módosítjuk egy gomb elnevezését, akkor a layout felborulhat, mivel az elemek egymáshoz viszonyított helyzete a nevük alapján tárolódik. Ezenkívül az alapértelmezett szövegformázási beállítások sem túl kényelmesek. Itt egy kicsit kódszintre menve tudni kell, hogy majd kódból nem hozhatunk fel a magunk által definiált osztályokat, ha már azokat eleve nem úgy definiáljuk az xml-ben. Itt például rádobunk egy View-t a fő nézetre, ami majd a mi saját ellenállás kijelzőnk lesz. Később, amikor kialakítjuk hozzá az osztályt át kell neveznünk az elemet saját típusunkra.




*29. Ami a tervezett felület mögött nyugszik*

Az elkészített erőforrásokból ezután a rendszer egy R fájlt, resource fájlt készít, ezen keresztül lesznek elérhetők a rendszer számára a menürendszer szövegei, és a kialakított felületetek. A programunkból már csak az így kapott publikus változókra kell hivatkoznunk, a

konkrét erőforrások elhelyezkedésével nem kell foglalkoznunk. A resource fájl-t nem szabad megváltoztatni, viszont hibás állapotba az editorokon keresztül is hozható. Ha valamilyen oknál fogva állapota nem megfelelő, törlésével bármikor kikényszeríthetjük újragenerálását. Az, hogy az elemek milyen néven szerepelnek ebben a fájlban az xml-ből derül ki ami generálja ezeket. A generált elemek a res mappa tartalmából állnak össze főképp, de a manifest fájlból is generálódhatnak elemek, mint például az alkalmazás neve, mint sztring erőforrás, onnét generálódik.

```
public final class R {
    public static final class array {
        public static final int band_color=0x7f040000;
        public static final int multiplier_color=0x7f040001;
        public static final int tolerance_color=0x7f040002;
    }
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int Band_1=0x7f060002;
        public static final int Band_2=0x7f060003;
        public static final int CalculateButton=0x7f060007;
```

*30. A rendszer által lefoglalalt erőforrások*

Az így elkészült erőforrásokat a MIDP-hez hasonló módon tudjuk majd a kész programba ágyazni. A különbség, hogy itt nem a kódban definiáljuk az elemeket, hanem a kész erőforrásokból összeszedjük és összekötjük azokat. Láthatóan a kód nem rövidebb a Mobile Edition által felkínált lehetőségnél, de a beállítások száma jóval nagyobb változatosságot kínál. Így belátható, hogy a megnövekedett funkcionalitást az Android hasonló komplexitással tudja kezelni.

A felhasználói interakció kezelésében továbbra is az eddig megismertekhez hasonló kód található. Az Android ugyan nem tartalmaz Command-hoz hasonló osztályt, de az esemény kiváltó hasonlóan paraméterekként adódik át, ennek következtében pedig közvetlen referencia összehasonlítást végezhetünk az események feldolgozása folyamán. A különbség újfent a

megjelenítés miatt keresendő, de a logikában fellelhető legnagyobb különbség nem értinti az Android-os API-t. Ez pedig annyi, hogy a sávoknak nem tartunk fent külön névvel ellátott változót, hanem csupán egy tömbben tároljuk azokat.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.setContentView(R.layout.main);
    band = new Spinner[4];
    band[0] = (Spinner) findViewById(R.id.Band_1);
    band[1] = (Spinner) findViewById(R.id.Band_2);
    band[2] = (Spinner) findViewById(R.id.Multiplier);
    band[3] = (Spinner) findViewById(R.id.Tolerance);
    calculateButton = (Button) findViewById(R.id.CalculateButt
    res_disp = (ResistorDisplay) findViewById(R.id.DrawSurface)
    res_disp.setVisibility(1);
    ArrayAdapter adapter = ArrayAdapter.createFromResource(this
            R.array.band_color, android.R.layout.simple_spinner
    adapter.setDropDownViewResource(android.R.layout.simple_spi
    band[0].setAdapter(adapter);
    band[1].setAdapter(adapter);
```

itt adjuk meg a layoutot

felszedjük a legördülő menüt

és hozzádrótozzuk az értékeket

*31. Az Activity életciklusának kezdete, inicializálás*

```
public void onClick(View v) {

    if (v == calculateButton){
        AlertDialog.Builder build
        builder.setMessage("Resis
public void onItemSelected(AdapterView<?> arg0, View arg1
        long arg3) {

    for(int i=0;i<4;i++){
        if(arg0 == band[i]){
            for(ResistorColor j : ResistorColor.values())
                if(arg0.getItemAtPosition(arg2).toString(
```

a parancsok feldolgozása nagyon hasonló

az iteratív megoldás az átgondolt design, mintsem az Androidnak köszönhető

*32. A user input feldolgozása hasonló a MIDP-hez*

A fő előnyt, a Mobile Edition-höz képest, az enumok felhasználása jelenti. Az enumok segítségével egy osztályban foglalhatóak a Mobile Edition-ben külön helyen letárolt ellenállás tulajdonságok. Ez a kód struktúráján sokat javított, de ezen felül biztonságosabbá és jobban olvashatóvá tette a programot.

```
public enum ResistorColor {
    BLACK("Black",0xFF000000,0,1,0),
    BROWN("Brown",0xFF400000,1,10,0.01),
    RED("Red",0xFFff0000,2,100,0.02),
    ORANGE("Orange",0xFFff8000,3,1000,0),
    YELLOW("Yellow",0xFF0000FF,4,10000,0),
    GREEN("Green",0xFF00FF00,5,100000,0.005),
    BLUE("Blue",0xFF0000FF,6,1000000,0.0025),
    VIOLET("Violet",0xFF9F5F9F,7,10000000,0.01),
    GRAY("Gray",0xFFC0C0C0,8,100000000,0.0005),
```

*33. Az enum osztályok hiánya nagyon érzékeny pontja a MIDP-nek*

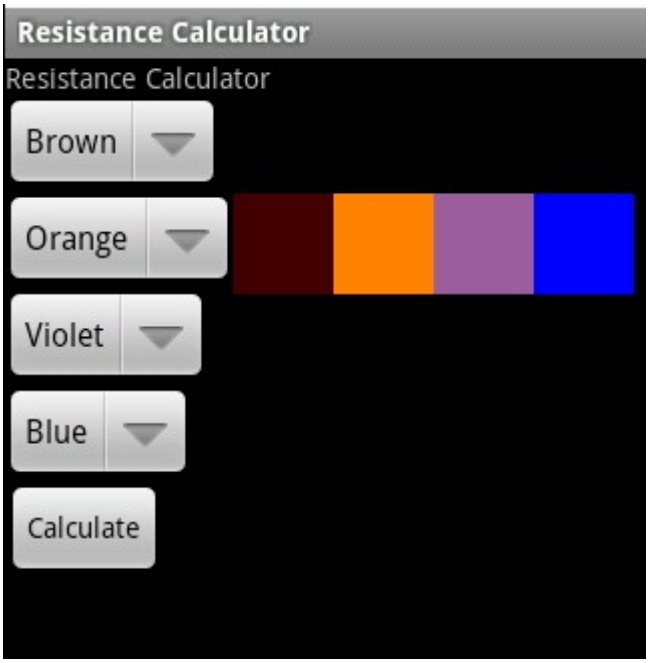
Az ellenállás megvalósítását végző Resistor osztályon látszódik, hogy az egyes függvények hossza jelentősen lerövidült. A Mobile Edition-ös megoldásom nagyon egyszerű volt, lehetett volna még rajta egyszerűsíteni, ennek következtében sok kódsorral járt, összesen 158-al. Ehhez képest az Android-os végső megoldásra sikerült ezt leszűkíteni 38 sorra. Feltételezhető, hogy egy olyan 100 kódsorra le lehetett volna szűkíteni a Mobile Edition-ös megoldást is, de abban az esetben új struktúrák bevezetésére lett volna szükség, és nem feltétlen a probléma megoldását szolgálta volna, mintsem újabbak előidézését. A ResistanceColor enum pedig jelentősen

```
 public ResistorColor getBand(int i) {
     return band[i];
 }
public void setBand(ResistorColor j, int band){
    this.band[band] = j;
}
public double getNameResistance(){
     return (band[0] .getBand()*10+band[1].getBand())*band[2
 }
public double getMinimumResistance(){
    return getNameResistance()*(1.0-band[3].getTolerance());
}
public double getMaximumResistance(){
    return getNameResistance()*(1.0+band[3].getTolerance());
}
```

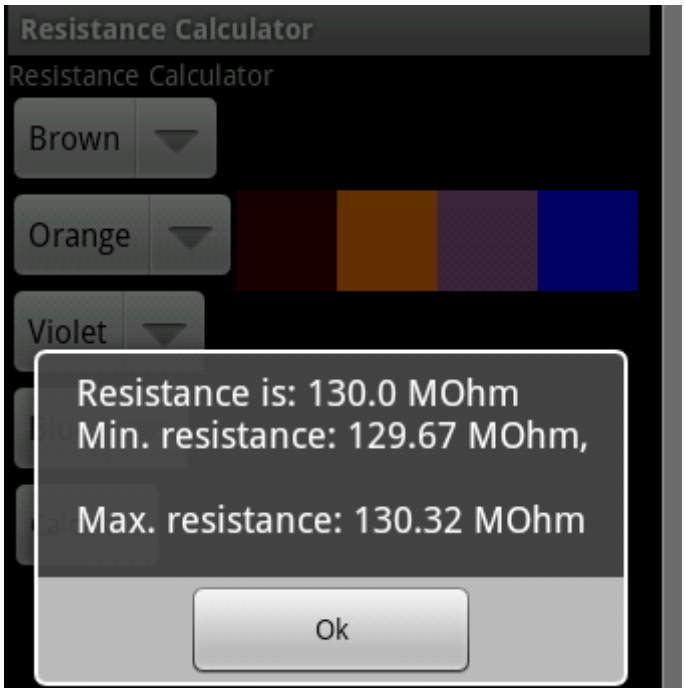
*34. Az enumok egy gyakorlati alkalmazása*

újrafelhasználható. Egy ilyen eszköz implementálása a projekt méretéhez képest aránytalanul nagy méretű erőfeszítést jelentett volna, így ez elmaradt.

A kész alkalmazás tesztelése emulátoron keresztül történik. A képeken megfigyelhetőek a nagy méretű gombok, amelyek az érintőképernyős használat kényelmességéért felelnek.




*35. A kész alkalmazás*



*36. Az eredmény megjelenítése*

Maga a sávok megjelenítése nem valami látványos, de a projektben szerepe az, hogy megmutassa, hogy milyen dinamikus elemeket lehet létrehozni. Az emulátorok működése jobban tükrözi a valós platformokon való megjelenését az alkalmazásnak, mivel ez nem külső implementációkban, hanem a Google által kínált implementációban valósul meg.

### 4.11 A fejlesztés során szerzett tapasztalatok összegzése

Miután elkészült a portolás, itt az ideje, hogy összevessük ezt, az eredeti alkalmazásunkkal, és megnézzük, milyen következtetések vonhatóak le a két szoftverplatformot illetően. Nyilvánvalóan látszik, hogy a két rendszer nagyon különböző, mégis a fejlesztés során sok közösen felhasznált részt találtunk.

A két környezet nyelve a Java. Bár a verzió különbözik, és ezáltal a nyújtott funkciók is, de a két alkalmazás alapvető elemeiben ugyan olyan. Az alkalmazás során voltak osztályok amelyekben semmilyen más osztályt nem importáltam, ezek az osztályok egy az egyben felhasználhatóak mind Android, mind Java Mobile Edition alatt, de a Standard Edition-ben is elfogadható osztályok lennének. A fejlettebb Android-os Java verzió miatt azonban mégis

ajánlatos volt testre szabni bizonyos részeket.

Az alkalmazás fő szerkezetén nem kellett változtatni, még az interfészek is hasonlóak voltak. Ezen túlmenően bár a metódusok elnevezése különbözött, mégis elmondható, hogy a grafikus megjelenítésre külön osztály van, amely rendelkezik alapvető rajzoló metódusokkal és szöveg kiiratással. A GUI elemek hasonlóak, de az, hogy az Android ezen téren XML alapú megkönnyíti az alkalmazás tervezését, illetve az XML-ben végrehajtott változtatások nem igénylik a kód újrafordítását.

A felhasznált IDE-k nagy segítséget nyújtanak az alkalmazás fejlesztésben, különösképp nagy előny a NetBeans-be beépített játékfejlesztési segédlet, amely kis fejlesztőcsoportok munkáját jelentősen előbbre viheti. Az Android ugyan nem rendelkezik ilyen eszközökkel, de tudni kell, hogy vannak ingyenes, ámde kellőképp robusztus eszközök ilyen jellegű feladat végrehajtására. A MIDP által játékfejlesztésre nyújtott lehetőségek pedig csupán két dimenziós programok létrehozását szolgálják, a 3 dimenziós grafikához valamilyen OpenGL ES binding-ot kell használni, amellyel mind az Android, mind a MIDP rendelkezik.

Azt, hogy a MIDP 2.0, még ha 2002-es tervezésű is, nem mondanám, hogy elmarad az Anrdoidtól. A fő különbség abban áll, hogy az Android egy sokkal határozottabb platform az alkalmazások számára. Nem függ a különböző implementációktól, csupán attól, hogy az adott funkcióval a készülék rendelkezik-e. Programozók számára minél nagyobb az alkalmazás, annál kényelmesebb lehet az Android használata, viszont kis alkalmazásoknál az API jelentősen több kódsorral programozható, mint a MIDP. Ebben az irányban haladva azonban visszatérnénk oda, hogy a MIDP viszont nyelvi szinten hiányol olyan megoldásokat, amiket már a fent bemutatott alkalmazásban is láthattunk.

A végkifejlet pedig az, hogy az Android jóval modernebb, mint a MIDP 2.0, ami a kettő közti korkülönbséget tekintve elvárható, azonban a MIDP is el van látva egy olyan jó fejlesztői környezettel a NetBeans képében, aminek köszönhetően az alkalmazás fejlesztése nem lesz körülményesebb, mint Android alatt. Az egyik alkalmazáshoz elkészített design-t nagyban átvehetjük a másik alkalmazás fejlesztésénél, és az osztályok fejlesztése is történhet közösen. Az Android ott tesz előnyre szert, hogy a közeg, amelyben ezek az alkalmazások futnak sokkal barátságosabbak. Az alkalmazások képesek interprocessz kommunikációra, a felépítésük

modulárisabb, a fejlesztő számára nagyobb szabadságot nyújtanak. A MIDP 3.0-ás szabvány is efelé törekszik, de az még majd csak ebben az évben fog eldőlni, hogy mire is jut ilyen szinten.

Végső soron a programozónak nem ott kell mérlegelni, hogy mely technológia az, melyben kényelmesebben tudja megírni a saját kódját, hanem melyik az, amelyen jobban eltudja érni a kívánt célcsoportot, és melyik az, amelyik a saját fejlesztésre fordított időt minimalizálja. Jelen pillanatban ebben az Android van előrébb.

## 5. Összegzés

Szakdolgozatom elején kiindultunk abból, hogy a mobil eszközök világa sokkal szélesebb, mint a mobiltelefonoké. Láthattuk, hogy sok különböző hardveres környezet és sok szoftveres környezettel van dolgunk. Szót ejtettünk arról, hogy a fejlesztés ezen eszközökre mennyire függ a hardveres és a szoftveres környezettől, és hogy ez a környezet mennyiben befolyásolja a fejlesztő által használt segédeszközöket. Láthattuk a ma legnépszerűbb elterjedt technológiákat, majd kicsit a jövőre is kitekintést nyerhettünk. Ezek után pedig tüzetesebben is górcső alá vettünk két egymáshoz nagyon közeli technológiát.

Ez a két technológia Java alapú volt, és a legtöbb ma elsődleges mobil technológiával egyetemben a kliens oldalon vannak jelen. Ezek a technológiák pedig egyre nagyobbak és összetettebbek lesznek, abban a korszakban, amikor a mobil telefonos internetelérés ugrás szerűen nő, és ezzel egy időben az alkalmazások is szerverre kerülnek át. Ezen robusztus technológiák jelenléte így kérdésessé válik, de legalábbis mindenképp meghatározó részükké válik, hogy legalább annyira fontos, hogy milyen módon tudnak webservice-eket elérni, mint az, hogy a felületük mennyire kényelmesen használható. Az API-k és a nyelvek hasonlóak. A mobil fejlesztésben C alapú nyelvek dominálnak, és a modern keretrendszerek hasonló elvek alapján fejlődtek az utóbbi időben. Ezeknél fogva nem igazán a programozó kényeztetésében rejlik egy platform sikere, hisz erre már hosszú ideje törekednek a gyártók, hanem abban, hogy a fejlesztő milyen csatornán jut el a felhasználóhoz, és hogy milyen élményt tud nyújtani a felhasználó számára. Az Oracle/Sun abszolút szabadsága sajnos egyikre sem adja meg a választ, míg az Apple -féle szigor valószínűleg nem vezethet teljes piaci dominanciához. E kettő között helyezkedik el a Google és a Nokia. Míg a Google egy szoftveres megoldást kínál és több hardvereszközt támogat, addig a Nokia még a szoftveres eszközök közt is nagy mozgásteret

enged a felhasználóknak és a fejlesztőknek egyaránt.

A helyzet talán a 80-as évek személyi számítógép piacához hasonlítható, azzal a különbséggel, hogy a készülékek jóval elterjedtebbek és nem annyira kihasználtak, mint a személyi számítógép piacon, így a piaci versengés jóval hosszabb ideig eltart majd. Ilyen körülmények között pedig a fejlesztőknek nincs más lehetősége, mint minél több platformon jelen lenni.

**6. Irodalom jegyzék**


- Carol Hamer - Creating Mobile Games (Apress 2007)
- Java ME Technology - http://java.sun.com/javame/technology/index.jsp
- Android Developers - http://developer.android.com/index.html
- Qt Developers - http://qt.nokia.com/developer
- Windows Phone - http://www.windowsphone7.com/
- Microsoft Kin - http://gizmodo.com/5531082/
- iPone Dev Center - http://developer.apple.com/iphone/index.action
- Maemo Development - http://maemo.org/development/
- Symbian Developer - http://developer.symbian.org/
- Cloud Computing and Developing Nations by Samuel Greengard - http://cacm.acm.org/magazines/2010/5/87255-cloud-computing-and-developing-nations/fulltext
- Who needs flash? By Scott Gilbertson- http://www.webmonkey.com/2010/05/who-needs-flash/
- Apple refuses pulitzer winners by Rob Pegoraro- http://voices.washingtonpost.com/fasterforward/2010/04/apple_refuses_pulitzer_winners.html
- Android Shakes up US Smartphone Market press release - http://www.npd.com/press/releases/press_100510.html
- Shifting fortunes for Cell Phones by Robert Cyran- http://www.nytimes.com/2010/05/12/business/12views.html
- Nokia and Intel plan new mobile platform by Andrew Parker - http://www.ft.com/cms/s/0/536f1ac6-1a66-11df-a2e3-00144feab49a.html
- Can Microsoft catch up on mobile? By Peter Burrows- http://www.businessweek.com/technology/content/feb2009/tc2009029_908364.htm

# Köszönetnyilvánítás

Ezúton szeretnék köszönetet nyilvánítani édesapámnak és édesanyámnak,hogy tanulmányaimban mindig támogatnak, szeretetük és gondoskodásuk nélkül idáig nem juthattam volna. Ezen kívül szeretnék köszönetet mondani Andrejcsik Tamásnak, Nagy István Zoltánnak, Nagy Ottónak, Tanyi Attilának az egyetem során nyújtott támaszukért. Legutolsó sorban pedig az Ensemble Studios egykori csapatának, akik efelé a pálya felé indítottak még gyerekkoromban.