Antti Leukkunen

JAVA-OHJELMOINNIN ERITYISPIIRTEET ANDROID-POHJAISISSA LAITTEISSA

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2012

TIIVISTELMÄ

Leukkunen, Antti
Java-ohjelmoinnin erityispiirteet Android-pohjaisissa laitteissa
Jyväskylä: Jyväskylän yliopisto, 2012, 28 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Hirvonen, Pertti

Android-käyttöjärjestelmä tarjoaa Java-ohjelmoijalle uusia haasteita sekä mahdollisuuksia. Androidiin siirtyminen vaatii kuitenkin joidenkin asioiden huomioon ottamista. Tässä tutkielmassa tarkastellaan Java-ohjelmoinnin erityispiirteitä, kun kohdelaitteina ovat Android-pohjaiset mobiililaitteet. Tutkielmassa tarkastellaan ensinnäkin Android-ohjelmoinnin perusperiaatteita ja verrataan niitä perinteiseen Java-ohjelmointiin. Erityispiirteet jaetaan kahdelle tasolle: ohjelmisto- sekä laitetasolle. Tutkimuksen tarkoituksena on selvittää, millä tavalla Java-ohjelmointi eroaa Android-ohjelmoinnista ja toisin sanoen, minkälaisia haasteita tai toisaalta mahdollisuuksia Android-maailmaan tutustuva Java-ohjelmoija tulee kohtaamaan.

Tämän tutkimuksen pohjana käytettiin alan kirjallisuutta sekä erilaisia julkaisuja aihepiirin alueelta. Aiheen ollessa verrattain nuori, käytettiin lähteinä myös internet-artikkeleita sekä Android-kehittäjien tuottamaa sivustoa.

Keskeisimpinä tuloksina tutkimuksessa nousivat esiin sovelluksien elinkaarien erilaisuus, järjestelmäpinon vaikutukset ohjelmakoodin kääntämiseen sekä Androidin epästandardi luokkakokoelma. Android-sovelluksen elinkaari määräytyy käyttäjän sekä muistinhallinnan toimien perusteella, kun taas perinteisessä Java-ohjelmoinnissa ohjelmoija hallitsee elinkaarta itse. Androidin epästandardista luokkakokoelmasta puuttuu esimerkiksi grafiikkakirjastot, joiden puute pakottaa sovelluskehittäjän tutustumaan uusiin toimintatapoihin.

Asiasanat: android, java, ohjelmistokehitys, erityispiirteet

ABSTRACT

Leukkunen, Antti
Specific features of Java-programming on Android-based devices
Jyväskylä: University of Jyväskylä, 2012, 28 p.
Information Systems Science, Bachelor's Thesis
Supervisor: Hirvonen, Pertti

Android operating system offers new challenges and opportunities for traditional Java-programmer. However there are some things which need to be taken in account when shifting to Android. This thesis views special features of Java-programming when the target device is Android-based. The beginning of the thesis introduces the basic principles of Android-programming which are compared to the traditional Java-programming. Special features are divided into two different parts: software level and hardware level. The purpose of the thesis is to clarify how traditional Java-programming differs from Android-programming. In other words what kinds of challenges or opportunities Java-programmer entering Android-world will encounter.

This study is based on a literature review as well as a variety of publications on the topic area. As the subject is relatively young, a variety of internet-sources and Android Developers -site were used as well.

The main results of the study were the differences in the application life cycle, the system stack effects on compilation of the source code and non-standard collection of classes. Android application life cycle is highly affected by the user's activity and by the actions of memory management while the traditional Java programmer controls the life cycle himself. The non-standard collection of classes lacks, for example, the graphic libraries which enforce the developer to explore the topic.

Keywords: android, java, software development, specific features

KUVIOT

SISÄLLYS

1 JOHDANTO

Android on mobiililaitteille sekä tablet-laitteille tarkoitettu ohjelmistopino, joka käsittää käyttöjärjestelmän, väliohjelmistoja sekä joitakin perusohjelmia. Sen on kehittänyt Open Handset Alliance (OHA), jota hallinnoi Google yhdessä monien laitevalmistajien kanssa. Androidiin kirjoitetaan Java-kieltä ja siinä käytetään Googlen kehittämiä Java-kirjastoja. Mobiililaitteiden käyttöjärjestelmät ohjaavat laitteiden toimintaa samalla tavalla kuin Windows tai Mac OS-käyttöjärjestelmä. Eroavaisuus on käytetyissä sovelluksissa, jotka ovat kevyitä ja yksinkertaisia. Älypuhelimien tulee hallita mobiiliverkkoja, multimediaa sekä erilaisia syötetekniikoita. Älypuhelimilla ja erityisesti Android-käyttöjärjestelmää hyödyntävillä laitteilla katsotaan olevan seuraavankaltaisia ominaisuuksia:

- Kannettavuus ja pieni koko
- Samanaikaiset ja jatkuvat yhteydet
- Laaja tuotevalikoima
- Perustuvuus avoimeen lähdekoodiin
- Rajoitetut resurssit

Älypuhelinmarkkinoiden laajassa valikoimassa on käytössä muutama merkittävä käyttöjärjestelmä: Nokian Symbian, Applen IOS, RIM:n BlackBerry OS, Microsoftin Windows-käyttöjärjestelmä sekä Googlen Android. (Shanker & Lal, 2011.)

Android-laitteiden suuri viimeaikainen suosio on johtanut niille sopivien sovellusten määrän kasvuun. Java-kielen omaksumisen helppous ja ohjelmien luomisen yleinen yksinkertaisuus ovat Android-ohjelmistokehityksen suosion kantavia tekijöitä. Avoimeen lähdekoodiin perustuvalla Androidilla onkin tämän hetken suurin kehittäjäkanta (Gavalas & Economou, 2011).

Java-kieli on tällä hetkellä suosituin ohjelmointikieli (Tiobe, 2012). Java on Sun Microsystemsin kehittämä oliopohjainen ohjelmointikieli ja Java-perheeseen kuuluvat myös luokkakirjastot sekä ajonaikaiset virtuaalikoneet. Kieli on yleiskäyttöinen ja luokkapohjainen ohjelmointikieli, joka suunniteltiin siten, että ohjelmakoodia voitaisiin suorittaa mahdollisimman laajasti laitealus-

tasta riippumatta. Javan sanotaan ottaneen vaikutteita C-kielen parhaista puolista. Muita merkittäviä piirteitä Java-kielelle ovat vahva tyypitys, automaattinen roskienkeruu sekä monipuoliset kirjastot. (Wikla, 2005b.)

Java-ohjelmakoodin kirjoittaminen Androidille kuitenkin poikkeaa joiltakin osin perinteisestä ohjelmoinnista ja joitakin asioita pitää ottaa huomioon. Android-laitekanta on hyvin laaja ja spesifikaatiot laitteiden välillä vaihtelevat suuresti: prosessori, muistit ja näytön koko ovat monesti merkittäviä tekijöitä sovelluksen toteutusvaiheessa.

Androidin avoimuus saattaa myös aiheuttaa ongelmia, sillä sovelluksilla voi olla vaikeuksia toimia vakaasti eri ohjelmistoversioiden (1.0 – 2.3) välillä. (Gavalas & Economou, 2011.) Android-kehityspaketin (SDK) ensimmäinen versio julkaistiin vuonna 2008 ilman, että yhtäkään laitetta oli vielä edes markkinoilla. 2009 julkaistiin versiot 1.5–2.1 ja 2010 päästiin jo versioon 2.2 asti. (Gargenta, 2011, s. 3.) Nopean kehityksen tuloksena on laitekanta, joka on pirstoutunut laajasti eri versioiden välille. Tämä aiheuttaa sen, että sovelluksia kehitettäessä täytyy ottaa huomioon vanhempia versioita käyttävät laitteet, mutta myös uusinta versiota edustavat Android-laitteet.

Tämän tutkimuksen tarkoituksena on selvittää, millä tavalla perinteinen Java-ohjelmointi eroaa Android-ohjelmoinnista. Tarkoituksena on kerätä erilaisia erityispiirteitä ja selvittää niiden pääkohtia sekä antaa mahdollisesti tietoa siitä, miten erityispiirteitä tulee lähestyä. Tutkimusongelma on jaettu karkeasti ohjelmistotason erityispiirteisiin sekä laitetason erityispiirteisiin. Aluksi tutustutaan Android-ohjelmointiin yleistasolla ja esitellään tyypillinen Hello World –tyyppinen sovellus. Seuraavissa luvuissa perehdytään laitetason sekä ohjelmistotason erityispiirteisiin. Luvussa kuusi verrataan Java-ohjelmointia Android-ohjelmointiin ja nostetaan esille merkittävimpiä eroavaisuuksia. Lopuksi tehdään yhteenveto ja pohditaan tutkielmassa esille nousseita seikkoja sekä otetaan kantaa mahdollisiin jatkotutkimusaiheisiin.

2 ANDROID-OHJELMOINNISTA

Android-sovelluskehitystä voidaan suorittaa yleisimmillä käyttöjärjestelmillä, joten ohjelmointi on mahdollista melkeinpä jokaisella modernilla tietokoneella. Ohjelmoija voi myös luoda erilaisia virtuaalisia Android-laitteita (AVD, Android Virtual Device), jotka vastaavat fyysisiä laitteita ominaisuuksiltaan. Virtuaalilaitteelle voidaan asettaa mm. haluttu näytön resoluutio, muistin määrä sekä tietty Android-versio, jolloin voidaan hyvin kattavasti testata sovellusta erilaisilla laitteilla ilman, että meillä on välttämättä ollenkaan fyysistä laitetta. Sovellusten testaaminen fyysisellä laitteella on kuitenkin korvaamatonta. (Goadrich & Rogers, 2011.)

Suosituimpana kehitysalustana on Eclipse, jota suositellaan yleisesti aloittelijoille ja sitä käytetään paljon korkeakoulutason opetuksessa. Eclipse ei tietenkään ole ainoa vaihtoehto, mutta se tarjoaa työkalut koko sovelluksen elinkaaren ajaksi aina ohjelmoinnista julkaisuun ja se on siten helppo valinta. Eclipselle on saatavilla erityinen Android-kehityspaketti, jonka avulla voi helposti luoda projekteja, ajaa niitä virtuaalilaitteilla sekä lopulta allekirjoittaa sovelluksia julkaistavaksi Android Marketissa. (Goadrich & Rogers, 2011.)

Android-sovelluksen graafinen ulkoasu koostuu XML-tiedostoista, jotka sisältävät erilaisia graafisia elementtejä sijoiteltuna erityisiin pohja-asetelmiin (layout). Eclipse sisältää yksinkertaisen ja toimivan graafisen editorin, jolla voidaan rakentaa ulkoasu juuri sellaiseksi kuin sen halutaan loppulaitteella näyttävän.

Android-projekti sisältää kolme pääelementtiä: Android-manifesti, erilaiset resurssit sekä aktiviteettitiedostot luokkineen. Seuraavissa alaluvuissa kuvataan miten perinteinen, Hello World –tyyppinen, sovellus toteutetaan Androidille.

2.1 Manifesti määrittelee perustiedot

Jokainen Android-sovellus tarvitsee AndroidManifest.xml–tiedoston, joka sisältää sovelluksen perustiedot, SDK-vaatimukset sekä mahdolliset erityisluvat, joita sovellus tarvitsee toimiakseen. Perustietoihin sisältyy sovelluksen nimi ja versionumerointi. Erilaisiin resursseihin, kuten kuviin tai esimääriteltyihin teksteihin viitataan @-merkillä. (Goadrich & Rogers, 2011.)

Manifestissä määritellään myös, miten sovellus sitoutuu muuhun järjestelmään. Esimerkiksi täytyy päättää, minkä aktiviteetin haluamme käynnistyvän, kun käyttäjä valitsee sovelluksemme kuvakkeen valikosta. Kun Eclipsessä luodaan Android-projekti, luodaan myös valmiiksi manifestipohja. Yksinkertaista sovellusta luodessa manifestiin ei juuri tarvitse tehdä muutoksia. Toisaalta manifesti voi kasvaa jopa satojen rivien mittaiseksi. (Allen, 2012, s. 33-34.)

Esimerkki 1. Näkymä Android-manifest.xml-tiedoston sisällöstä. Sisältö määrittää mm. Androidin pienimmän mahdollisen versionumeron, sovelluksen nimen sekä sovelluksen käyttämät aktiviteetit.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.esimerkki.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="7" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".HelloAndroid"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent
                .category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

2.2 Sovelluksen vaatimat resurssit

Resursseiksi luetaan erilaiset bittikartat eli yksinkertaisemmin kuvat ja merkkijonot. Myös graafista ulkoasua varten tarvittavat XML-tiedostot luetaan tähän kategoriaan. Erilliset merkkijonoluettelot ovat tarvittavia, kun luodaan käännöksiä toisille kielille. Tällöin luodaan kansioita, jotka ovat nimetty kielen mukaan: values-fi suomenkieltä, values-hi hindiä varten jne. Values-kansiota ilman päätettä käytetään merkitsemään sovelluksen oletuskieltä. Android järjestelmä

käyttää ensisijaisesti myös järjestelmän oletuskieltä vastaavia arvoja, muutoin values-kansiossa sijaitsevia sovelluksen oletusarvoja. (Goadrich & Rogers, 2011.)

Resurssit ja Java yhdistetään R.java-tiedostolla, joka pitää sisällään viitteet kaikkiin käytettyihin resursseihin, kuten merkkijonoihin ja kuviin. R-tiedosto luodaan automaattisesti ja se päivittyy aina kun resurssitietoja lisätään tai muokataan. Esimerkiksi viittaus pääulkoasuun merkittäisiin Java-koodiin R.layout.main. (Gargenta, 2011, s. 22.)

Esimerkki 2. Esimerkissä on kuvattu /res/values/strings.xml-tiedoston sisältö, johon on kirjoitettu sovelluksen nimi sekä toinen merkkijonoarvo, johon voidaan viitata Java-koodissa R.string.hello-viitteellä.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello Android!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

Graafinen ulkoasu määritellään /res/layout/-kansiossa yleensä main.xml-tiedostossa. Allaolevasssa ulkoasumäärittelyssä on käytetty suhteellista ulkoasutyyppiä. Ulkoasuun on myös määritetty keskitetty TextView-näkymä, jonka yksilöllinen viite on @+id/TextView01 ja sisällön viite @string/hello. Määrittelyt voivat sisältää mm. marginaaleja, fonttimäärittelyjä, tekstin tasauksia sekä tekstisisällön viitteitä. (Goadrich & Rogers, 2011.)

Esimerkki 3. Main.xml-ulkoasutiedoston sisältö. Sisältö määrittelee ensimmäiseksi, että käytössä on lineaarinen ulkoasumalli, jonka tuottama näkymä on pystysuora. Lineaarinen ulkoasumalli sisältää yhden tekstinäkymän, joka saa sisältönsä esimerkissä 2. mainitusta ”hello”-nimisestä merkkijonosta.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
</LinearLayout>
```

2.3 Aktiviteetti sovelluksen perusyksikkönä

Aktiviteetti (Activity) on yksittäinen sovelluksen osa, johon keskitytään kerrallaan. Suurin osa aktiviteeteistä kommunikoi käyttäjän kanssa ja toisinpäin. Ak-

tiviteetti yleensä luo näytölle ikkunan, johon oma ulkoasunäkymä sijoitetaan. Aktiviteetti käsitetään usein koko ruudun pinta-alan peittävänä näkymänä, mutta se voi myös olla kelluva ikkuna tai vaikka toiseen aktiviteettiin upotettuna. Ensimmäinen metodi, joka löytyy melkeinpä kaikista aktiviteeteista, on onCreate(). Kyseisessä metodissa kutsutaan tavallisesti ulkoasurakentajaa ja luodaan yhteydet ulkoasun eri osiin hakemalla ne findViewById()-metodia käyttäen. Toinen tavanomainen metodi on onPause(), jossa kerrotaan mitä tapahtuu, kun käyttäjä poistuu aktiviteetista sammuttamatta sitä kokonaan. Yleensä kaikenlaiset muutokset ja asetukset tallennetaan tässä vaiheessa, jotta tietoa ei häviä. Aktiviteetin elinkaareen kuuluvat myös olennaisesti onDestroy(), jota kutsutaan, kun aktiviteetti lopetetaan kokonaan. (Gargenta, 2011, s. 28–29.)

Esimerkki 4. Pääaktiviteetin sisältämä Java-ohjelmakoodi. Tämä aktiviteetti sisältyy pakkaukseen com.esimerkki.helloandroid. Pakkaukset määrittelevät Java-luokkien välisiä näkyvyyksiä sekä niillä on myös tärkeä tehtävä sovellusten yksilöinnissä. HelloAndroid-aktiviteetti periytyy Activity-luokasta. Määriteltynä on myös onCreate-metodi, jossa ainoastaan asetetaan aktiviteetin käyttämäksi ulkoasumalliksi Main.

```
package com.esimerkki.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

2.4 Java-ohjelmointi ja Hello World

Perinteinen Java-ohjelmointi eroaa jonkin verran pääpiirteiltään Android-ohjelmoinnista. Kuten edellä havaitsimme, yksinkertainenkin Android-sovellus vaatii useita eri lähdetiedostoja, jotta sovellus voidaan ajaa päätelaitteessa. Java-ohjelmoinnissa tarvitaan vain yksinkertainen Java-lähdekoodia sisältävä tiedosto, johon tarvittavat käskyt on kirjoitettu.

Esimerkit eivät ole suoraan vertailukelpoisia keskenään, sillä esimerkissä 5 merkkijono tulostetaan konsoliin eikä graafiseen elementtiin, kuten Androidin vastaavassa. Jotta voisimme avata näytölle tekstikenttiä ja muita graafisia elementtejä, täytyisi Java-ohjelmakoodissa käyttää esimerkiksi AWT-luokkaa. Androidin esimerkkiä voitaisiin muokata vastaamaan enemmän Javaa siten, että tulostaisimmekin merkkijonon tekstikentän sijaan lokiin käyttämällä Log-

luokkaa. Tärkeintä on kuitenkin kiinnittää huomiota ohjelmien perusvaatimusten erilaisuuteen, eikä niinkään itse Hello World –tulosteeseen.

Esimerkki 5. Java-ohjelmoinnin mukainen Hello World –ohjelma, joka tulostaa tekstin ”Hello World” konsoliin. Ohjelma sisältää Java-kielelle ominaisen pääohjelman eli main-metodin, joka suoritetaan kun ohjelma ajetaan. Pääohjelmassa käytetään System-luokkaa tulostamaan haluttu merkkijono konsoliin.

```
public class HelloWorld {

           public static void main(String[] args) {
                      System.out.println("Hello World");
           }
}
```

3 LAITETASON ERITYISPIIRTEET

Laitetasolla tarkoitetaan Android-laitteiden fyysisiä komponentteja. Nykyinen laitekanta on hyvin laaja ja siten myös suorituskyky vaihtelee älypuhelinten ja tablet-laitteiden välillä erittäinkin paljon. Suorittimet, näytönohjainpiirit, näytöt sekä muistin määrä ovat merkittäviä muuttujia, kun pohditaan esimerkiksi uuden laitteen hankintaa, mutta toisaalta myös niille tapahtuvaa sovelluskehitystä. Verrattuna Applen iOS-käyttöjärjestelmää käyttäviin laitteisiin on Android-laitteiden kirjo huomattavasti laajempi: iPhoneja, iPadeja ja iPodeja on alle kaksikymmentä, kun taas Androidilla puhutaan sadoista laitteista.

Kun ensimmäinen Android-laite T-Mobile G1 julkaistiin, oli helppoa määritellä millaisia resursseja meillä on käytössä. G1:n tapauksessa voitiin ajatella, että käytössä on mm. fyysinen näppäimistö sekä ohjauspallo navigointia varten. Nykyään on tavallisempaa, että fyysistä näppäimistöä ei ole, mutta sen sijasta laitteessa voi olla peliohjaimen tapaisia painikkeita. On myös mahdollista, että laitteessa on kaksi näyttöä. (Allen, 2012, s. 551.)

Ohjelmointialustan monimuotoisuudella on hyvät ja huonot puolensa. Sovelluskehittäjän täytyy ottaa huomioon suorituskykyjen eroavaisuudet ja suunnitella sovellus siten, että se on riittävän yhteensopiva laitekantaan nähden. Paljon suoritintehoa vaativa sovellus ei tietenkään ole käyttökelpoinen vanhemmissa tai edullisemmissa malleissa. Suoritintehon hyödyntämättä jättäminenkään ei ole järkevä vaihtoehto.

Pöytätietokonetta ja mobiililaitetta verrattaessa keskenään nousee nopeasti esille tiettyjä rajoitteita, joita täytyy ottaa huomioon. Meier on listannut niistä merkittävämmät seuraavasti:

- Vaatimus tehokkuudelle
- Keskus- ja massamuistien rajallisuus
- Vaihtelevat näyttökoot
- Tiedonsiirron viiveet sekä rajallinen nopeus

Jokainen uusi laitesukupolvi helpottaa näitä edellä mainittuja piirteitä, mutta se ei poista markkinoilta vanhaa laitekantaa, joka täytyy ottaa sovelluskehityksessä huomioon. Hyvänä käytäntönä Meier kehottaa varautumaan vaikeinta mahdollista tilannetta vastaan. (Meier, 2010, s. 30–31.)

3.1 Tehokkuusvaatimukset

Laitevalmistajat ovat pyrkineet perinteisesti tekemään laitteista pieniä sekä arvostamaan pitkää akun käyttöaikaa. Nykyään asia on hieman toisin. Näyttökoot kasvavat ja akun käyttöikä pienenee käänteisessä suhteessa näytön koon sekä muiden virtaa vievien ominaisuuksien määrän kasvaessa. Toki akutkin kehittyvät, mutta omaa tahtiaan. Mooren lain mukaan transistorien lukumäärä kaksinkertaistuu mikropiireissä joka toinen vuosi. Mobiililaitteilla se käytännössä tarkoittaa sitä, että ohjelmakoodia täytyy optimoida, jotta sitä voidaan ajaa nopeasti ja tehokkaasti. Tehokkuus on erittäin tärkeää laitteissa, joissa resurssit ovat rajalliset. Kuten Meierkin tekstissään kertoo, tehokkuus on aihealueena hyvin laaja, eikä sitä pyritä tässä tarkasti kuvaamaan. (Meier, 2010, s. 31.)

3.2 Massa- ja keskusmuisti

Flash-muistit sekä SSD-levyt ovat kehittyneet lähivuosina merkittävästi ja siten myös mobiililaitteiden massamuistikapasiteetit ovat kasvaneet. Sisäiset muistit ja muistikortit yhdessä mahdollistavat mobiililaitteelle helposti 20 gigatavua massamuistikapasiteettia. Toisaalta perinteiset hyötysovellukset eivät ole niitä suurimpia massamuistien käyttäjiä vaan musiikki, kuvat, videot sekä pelit ovat merkittävimpiä tilanviejiä. Toukokuussa 2010 Androidin 2.2 version, ”Froyon”, yhteydessä julkistettiin muutos, joka mahdollisti sovelluksien asentamisen ulkoiseen muistiin. Tuolloin sisäisen muistin riittävyyttä piti tarkkailla, eikä ylimääräisiä sovelluksia yksinkertaisesti mahtunut muistiin. Mahdollisuus käyttää ulkoista muistia osaltaan vapautti tilannetta. Tämä ei silti tarkoita, ettei sovelluksen koosta tarvitsisi välittää. Markkinoilla on yhä laitteita, jotka käyttävät Androidin aiempia versioita sekä kuten aiemmin totesimme, massamuistilla on muitakin käyttötarkoituksia.

Suurin huolenaihe ei yleensä ole käännetyn sovelluksen koko itsessään, vaan sovelluksen käytön vaatimat resurssit. Esimerkiksi sähköisten kirjojen lukemiseen tarkoitettu sovellus ei välttämättä itsessään ole kovinkaan merkittävän kokoinen, mutta sen sisältönä toimivat kirjat sen sijaan täytyy sijoittaa jonnekin. Android tarjoaa muutamia erilaisia tapoja tallettaa tietoa. Suurille tietomäärille on Androidin tietokantoja sekä sisällöntuottajia, kun taas pienille tietomäärille on olemassa oma optimoitu kehys. Myös itse tiedostojärjestelmään kirjoittaminen on mahdollista, mutta silloin pitää miettiä, miten toteuttaa teho-

kas tiedostorakenne. Välimuistin käyttö voi olla tehokasta varsinkin, kun tiedonsiirto ei ole niin tehokasta kuin perinteisissä tietokoneissa. Tiedostot ja rekisteri on muistettava myös hävittää järjestelmästä, kun niille ei enää ole käyttöä. (Meier, 2010, s. 31–32.)

3.3 Näyttöjen monimuotoisuus

Android-järjestelmä ei itsessään tee oletuksia päätelaitteen komponenteista ja se suunniteltiinkin toimimaan hyvin monenlaisissa laitteissa (Gargenta, 2011, s. 2). Näytön rajallinen koko on haastava tekijä suunniteltaessa hyviä, käyttäjäystävällisiä käyttöliittymiä. Meier suosittelee, että käyttöliittymä rakennetaan siten, että kaikki tarpeellinen tieto voidaan havaita vain vilkaisemalla näyttöä: ”Make your applications intuitive and easy to use by reducing the number of controls and putting the most important information front and center. (2010, s. 32).” On tärkeää yrittää korvata suurta tekstimäärä ja painikkeita väreillä, kuvilla ja muodoilla, jotta tieto saadaan nopeasti ruudulta käyttäjälle. Samalla on syytä muistaa, että loppukäyttäjä kommunikoi laitteen kanssa pääosin kosketusnäytön kautta: interaktiivisten elementtien on oltava riittävän kokoisia, jotta niiden käyttäminen on sujuvaa. (Meier, 2010, s. 32.)

Android 1.0:n julkaisuvaiheessa kaikki Android-laitteet käyttivät samaa näyttökokoa ja –resoluutiota: HVGA, 320 x 480, 3.5”. Vuoden 2009 lopussa tilanne muuttui, kun laitekanta alkoi kasvaa voimakkaasti. Laajan laitekannan myötä myös näyttökoot vaihtelevat suuresti. (Allen, 2012, s. 271.) Näyttöjen resoluutioihin viitataan IBM:n esittelemän näyttöstandardin mukaisesti. Esimerkiksi Samsungin Galaxy Tab –laitteessa on 7’’:n WSVGA-näyttö, kun taas HTC:n Wildfiressä 3.2’’:n QVGA-näyttö. Hyvän ulkoasun takaamiseksi pitää huolehtia käyttöliittymän skaalautuvuudesta. (Meier, 2010, s. 32.)

3.4 Tiedonsiirron rajoitukset

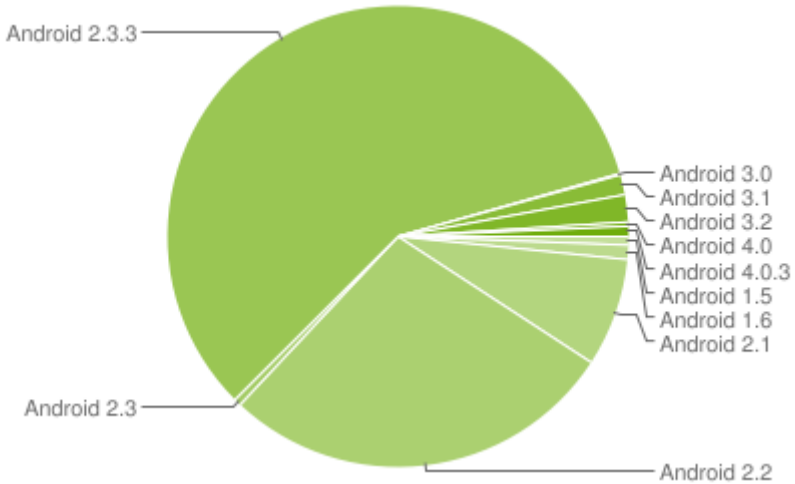
Älypuhelimet perustavat monet toiminnoistaan internet-yhteyden varaan ja ne käyttävät joko langattomia lähiverkkoyhteyksiä tai matkapuhelinverkkojen teknologioita. Rajoittavia tekijöitä tiedonsiirron osalta ovat nopeus, luotettavuus sekä saatavuus. Sovellusten täytyy myös pystyä käsittelemään tilanteita, joissa tietoa katoaa tai jää kokonaan löytymättä. Tilanne on paranemassa, kun 4G-verkot yleistyvät, kaupungit tarjoavat langattomia lähiverkkoja sekä teleoperaattorien tarjonta kasvaa kilpailun myötä. (Meier, 2010, s. 32.) Suomen kannalta tilanne on etenkin otettava huomioon, sillä verkon katvealueita tulee löytymään varmasti vielä pitkään. Tarkasteltaessa Soneran kuuluvuuskarttaa voidaan havaita, että Suomen pohjoisosissa ei ole täydellistä GSM-peittoa 3G-peiton ollessa vieläkin rajallisempi (Sonera, 2012). Meier (2010, s. 32) kehottaa

jälleen varautumaan vaikeimpaan mahdolliseen tilanteeseen, jotta käyttökokemus pysyy laadultaan korkeana.

4 OHJELMISTOTASON ERITYISPIIRTEET

Android perustuu avoimeen lähdekoodiin ja se kehittyy jatkuvasti nopeaa vauhtia. Käyttöjärjestelmän asettamat vaatimukset sovelluksille muuttuvat ja sovelluksia pitää ylläpitää jatkuvasti. Tätä kirjoittaessa Android-ohjelmistopinosta on julkaistu versio 4.0.4, joka kantaa nimeä Ice Cream Sandwich. Yli puolet käytössä olevista Android-laitteista käyttää yhä Gingerbread-versiota 2.3.3, kun versioita 3-4.0.4 käyttää alle 5-prosenttia (kuvio 1). Tämänkaltaisen laajan versiojakauman vuoksi puhutaan yleisesti Androidin pirstoutuneisuudesta.

Androidin kehittäjäryhmä on julkaissut paljon materiaalia sovelluskehitystä helpottamaan ja kehittäjien sivustolta löytyykin paljon ohjeita tehokkuuden parantamiseksi. Sivustolla on mainittu kaksi perussääntöä tehokkaan ohjelmakoodin kirjoittamiseen: ”Älä tee työtä, jota ei tarvitse tehdä” sekä ”Älä varaa muistia, jos sen voi välttää”. Nämä itsestäänselvyyksiltä kuulostavat ohjeet on hyvä muistaa, sillä monimutkaista ohjelmakoodia kirjoitettaessa perusasiat unohtuvat helposti. (Android Developers, 2012a.)

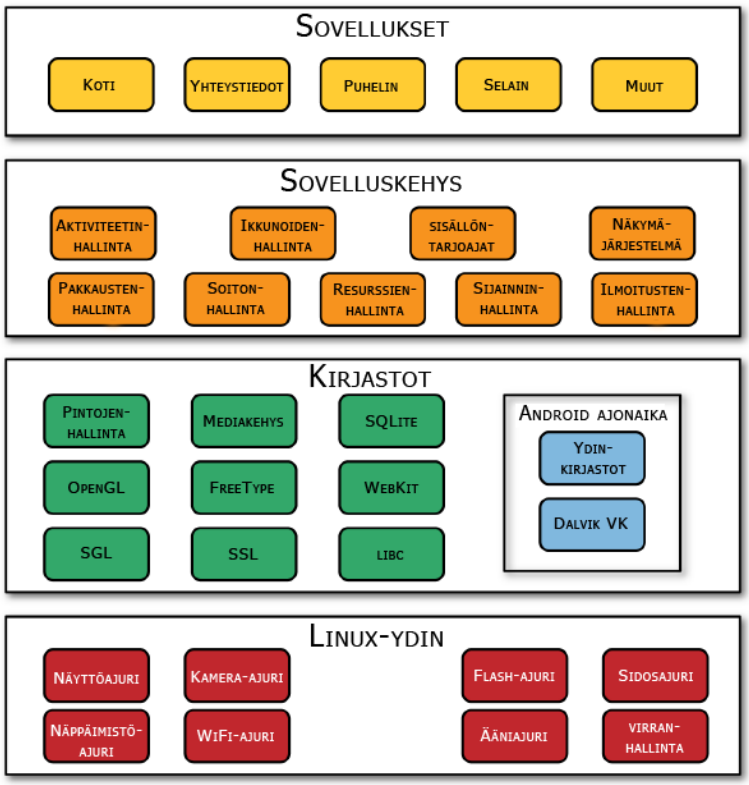


KUVIO 1 Android-versiojakauma (Android Developers, 2012b).

4.1 Android-ohjelmistopinon rakenne

Android ei ole pelkästään käyttöjärjestelmä, vaan se koostuu tasoista ja jokaisella tasolla on omanlaisensa piirteet sekä käyttötarkoitukset. Tasot eivät ole täydellisesti erillisiä vaan sulautuneet toisiinsa joissakin määrin. Kuviossa 2 on Androidin järjestelmäpino, josta käy ilmi eri tasot ja niiden sisältämät pääkomponentit. (Gargenta, 2011, s. 7.)

Rakenne koostuu neljästä tasosta, joissa on yhteensä viisi ryhmää. Päällimmäisellä tasolla eli sovellustasolla sijaitsee perustason sovelluksia kuten internet-selain, puhelin, yhteystiedot, SMS-viestit. Kaikki edellä mainitut sovellukset ovat kirjoitettu Java-kielellä. Huomattavaa on se, että Android-järjestelmä tukee moniajoa eli se mahdollistaa vaikkapa musiikin kuuntelemisen samaan aikaan, kun kirjoitetaan tekstiviestiä. Sovellustaso on se taso, jolla suurin osa käyttäjistä toimii. Seuraavana on sovelluskehyksen taso, joka määrittelee sovelluksen perusrakenteen. Kirjastotasolla taas on erilaisia kirjastoja liittyen esimerkiksi ääneen ja grafiikkaan. Kirjastot on kirjoitettu C/C++-kielellä, mutta niitä kutsutaan Javan käyttöliittymän avulla. Samalla tasolla on toinen ryhmä nimeltään Android-suoritusvaihe tai ajonaikaisuus. Ryhmä koostuu kahdesta komponentistä, joista toinen on ydinkirjastot ja toinen Dalvik-virtuaalikone, jota käsitellään tarkemmin luvussa 4.3. Android-järjestelmän pohjalla on Linux-ydin, jota käsitellään luvussa 4.2. (Speckmann, 2008.)



KUVIO 2 Android-käyttöjärjestelmän tasot ja pääkomponentit (Gargenta, 2011 s. 8.)

4.2 Linux-pohjaisuus Androidissa

Android-käyttöjärjestelmä on rakennettu Linux-ytimen päälle. Tärkeimpiä syitä Linux-pohjaisuuteen on siirrettävyys, turvallisuus sekä ominaisuudet liittyen muistin-, virran- ja verkonhallintaan. Linux-pohjaisuus antaa hyödyllistä laitteistoriippumattomuutta ja se tekee Androidista kokonaisvaltaisen. Android on tavallaan ohjelmistopino, joka voidaan sijoittaa huoletta erilaisten komponenttiarkkitehtuurien päälle. Linuxin pohjimmaisista osista monet on kirjoitettu C-kielellä, joka tekee Androidista käyttökelpoisen monissa laitteissa. (Gargenta, 2011, s. 7.)

Linux on myös erittäin turvallinen järjestelmä ja Android nojaa turvallisuudessaan siihen. Kaikki Androidin sovellukset ajetaan erillisinä Linux-prosesseina, joille Linux asettaa tiettyjä lupia (engl. permission). (Gargenta, 2011, s. 8.) Vaikka Android on rakennettu Linux-ytimen päälle, alustalla on hyvin vähän yhteisiä tekijöitä tavallisen pöytätietokoneissa käytettävän Linuxin kanssa. On siis merkittävää erottaa Linux ja Android toisistaan eli Android ei ole Linux. Ehkäpä suurin hyöty, jonka Linux tarjoaa Android-maailmaan, on korkean tason yhdenmukaisuus: periaatteessa suurimman osan Android-sovelluksista voi ajaa millä tahansa Android-laitteella. Toisaalta Androidin eristäytyneisyys aiheuttaa sen, että Linuxin laajaa sovelluskenttää ei voida hyödyntää Androidissa ollenkaan. Android ei virallisesti tue C-ohjelmia ollenkaan, joten GTK+ tai Qt-sovellusten kääntäminen Androidille ei onnistu. MIDP-sovelluksetkaan eivät sovi Androidiin, sillä virtuaalikone ei ole yhteensopiva. (Paul, 2009.)

Android käyttää Linux-ytimen versioita 2.6.x, jonka jälkimmäisin numero kasvaa aina uuden Android-version myötä. Linux-ydin sijoittuu siis heti laitetason ja ohjelmistotason väliin, jossa kommunikointi itse fyysisen laitteen ja ohjelmistojen välillä tapahtuu. Laitevalmistajien käyttäessä eri komponentteja laitteissaan myös Linux-ydin vaihtelee komponenttien mukaan. Linux-ydin on lisensoitu GNU:n julkisella lisenssillä, joten laitevalmistajien täytyy julkaista ytimen lähdekoodi samalla kun uusi laite julkaistaan. (Shanker & Lal, 2011.)

4.3 Dalvik-virtuaalikone

Google kehitti Androidia varten erityisen Dalvik-virtuaalikoneen. Javan virtuaalikone suunniteltiin alun perin kaikenlaisia tarpeita varten, mutta Dalvikin kehittäjäryhmä uskoi pystyvänsä kehittämään paremman virtuaalikoneen pelkästään mobiililaitteita varten. He kävivät läpi asioita, joiden he eivät uskoneet muuttuvan mobiililaitteiden kohdalla tulevina vuosina: akkujen valmiusaika sekä laitteiden laskuteho. Toinen merkittävä syy oman virtuaalikoneen kehittämiselle on se, että Javan virtuaalikone on lisensoitu toisin kuin itse Java-kieli ja sen työkalut. (Gargenta, 2011, s. 9-10.)

Dalvik on rekisteriperustainen virtuaalikone, joka on optimoitu siten, että laite voi ajaa montaa virtuaalikonetta yhtä aikaa. Dalvik tukeutuu Linux-ytimeen säikeistämisessä sekä alemman tason muistinhallinnassa. (Meier, 2010, s. 13.)

Dalvikin tehtävä on kääntää lähdekoodi Dalvikin ymmärtämään muotoon ja ajaa lopulta sovellus Dalvik-virtuaalikoneella. Perinteisessä Java-ohjelmoinnissa kirjoitetaan lähdekooditiedosto, joka käännetään Javan tavukoodiksi, jota puolestaan Javan virtuaalikone tulkkaa. Androidin kohdalla tarvitaan yksi välivaihe, joka sijoittuu tavukoodin ja virtuaalikoneen väliin: Javan tavukoodi käännetään vielä Dalvik-tavukoodiksi. Tämä Dalvik-tavukoodi sitten tulkataan Dalvik-virtuaalikoneella. Lisävaihe saattaa kuulostaa siltä, että tarvitaan enemmän työvaiheita. Kuitenkin kaikki käännöksen vaiheet on automatisoitu työkaluihin, kuten Eclipseen. (Gargenta, 2011, s. 10). Oletuksena jokainen Android-sovellus ajetaan erillisenä Dalvik-virtuaalikoneessa, jolloin muistin- ja prosessinhallinta suoritetaan ajon aikana (Meier, 2010, s. 57). Dalvik-tavukoodit kirjoitetaan .dex-tiedostoihin, jotka ovat Dalvikin omia suoritettavia tiedostoja. Dex-muoto on optimoitu käyttämään mahdollisimman vähän muistia.

Androidin kehittäjät olisivat voineet rakentaa järjestelmän siten, että Java-kieltä käännettäisiin suoraan Dalvik-tavukoodiksi, mutta Dalvikin haluttiin pohjautuvan Javan tavukoodiin, koska se on vakiintunutta eikä muutu versioiden myötä toisin kuin Java-kieli. Sivutuotteena Android-ohjelmia voidaan periaatteessa kirjoittaa millä tahansa kielellä, joka kääntyy Javan tavukoodiksi. (Gargenta, 2011, s. 11.)

5 JAVA-OHJELMOINTI JA ANDROID

Java-ohjelmoinnin ja Android-ohjelmoinnin välillä on joitakin eroavia tekijöitä. Perinteisessä Java-ohjelmoinnissa ohjelmoija luo oman maailmansa ja hallitsee sitä välittämättä muista ympärillä toimivista prosesseista sen enempää. Tavallisesti luodaan ohjelmalle pää-ikkuna ja sille erilaisia interaktiivisia osia, kuten tekstinsyöttämiseen tarvittavia kenttiä ja esimerkiksi painikkeita. Jos ohjelman tarvitsee keskustella muiden ohjelmistojen kanssa, se tehdään normaalisti erilaisten ohjelmarajapintojen kautta tai sitten tietokantojen avulla. Android-maailmassa toimitaan toisin. Rajattujen resurssien laitteissa täytyy hallita kokonaisuutta. Android-sovelluskehittäjän täytyy ottaa huomioon erilaiset tilanteet sovelluksen elinkaarta ajatellen: mitä tehdään, kun sovellus käynnistyy tai se tuhotaan, jotta muille sovelluksille saadaan vapautettua enemmän muistia käyttöön. (Allen, 2012, s. 4-5.)

5.1 Androidin epästandardi luokkakokoelma

Merkittävä eroavaisuus Javan ja Androidin välillä on se, että Android-ohjelmoinnissa käytetään epästandardia Javan luokkakokoelmaa. Javan merkittävimpiä ajoympäristöjä ovat Java SE, Java EE sekä Java ME. Androidin Java-kirjastot ovat lähinnä Javan SE-versiota, joka on tarkoitettu tavallisille työpöytätyyppisille ohjelmille. Suurin eroavaisuus on käyttöliittymäkirjastojen kuten AWT:n ja Swingin puute. Toisaalta Android lisää melko paljon omia kirjastojaan tuttujen SE-kirjastojen tueksi. (Gargenta, 2011, s. 11.)

Burnette (2008) on listannut Androidin sisältämiä ja siitä pois jätettyjä pakkauksia. Seuraavassa listassa on lueteltu pakkauksia, jotka löytyvät samannimisinä myös Androidin kokoelmasta.

- java.io - Syöte- ja tulostusvirrat
- java.lang – Kielet ja poikkeuksenhallinta

- java.math – Lukujenkäsittely, pyöristäminen
- java.net - Verkkojen käsittely, soketit
- java.nio – Uusi I/O
- java.security – Valtuuttaminen, sertifikaatit
- java.sql - Tietokantojen käyttöliittymät
- java.text – Tekstinkäsittely, luonnolliset kielet
- java.util – Listat, taulukot, kokoelmat
- javax.crypto – Tiedon salaaminen
- javax.net - Soketit
- javax.security – Todentaminen ja turvallisuus
- javax.sound – Musiikki ja äänet
- javax.sql – Tietokannat ja niiden hallinta
- javax.xml.parsers – XML-kielen jäsentimet
- org.w3c.dom – DOM-solmut ja elementit
- org.xml.sax – SAX-rajapinnan työkalut

Java-ohjelmoijan kannalta ehkä merkittävämpää on se, mitä Androidin Java ei pidä sisällään. Nämä eroavaisuudet ovat asioita, joita Androidiin tutustuvan Java-ohjelmoijan täytyy opetella tarpeen mukaan. Merkittävimpinä puuttuvina kirjastoina mainittakoon grafiikkakirjastot AWT ja Swing. Seuraavaksi on listattu Androidin kokoelman ulkopuolelle jätetyt kirjastot.

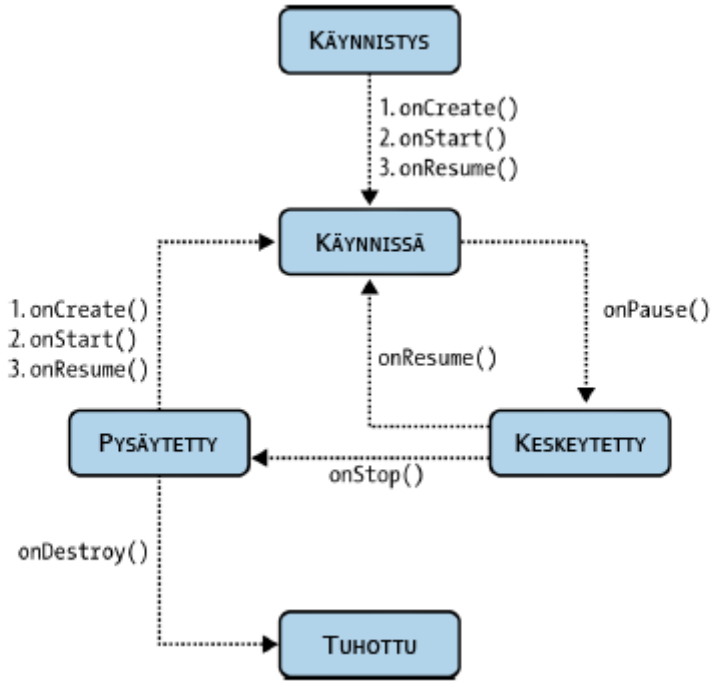
- java.applet
- java.awt
- java.beans
- java.lang.management
- java.rmi
- javax.accessibility
- javax.activity
- javax.imageio
- javax.management
- javax.naming
- javax.print
- javax.rmi
- javax.security.auth.kerberos
- javax.security.auth.spi
- javax.security.sasl
- javax.swing
- javax.transaction
- javax.xml (lukuun ottamatta .parsers-pakkausta)

5.2 Sovelluksen elinkaari

Android-sovelluksen ja Java-sovelluksen elinkaaressa on merkittäviä eroavaisuuksia. Android-sovelluksessa ei ole mitään tiettyä päämetodia, joka suoritetaan vaan sen sijaan sovellukselle määritelleen metodeja, jotka suoritetaan tarpeen mukaan. Java-ohjelmoinnissa tarvitaan main-metodi eli pääohjelmametodi, jotta jokin luokka voidaan suorittaa sovelluksena. Androidiin verrattuna elinkaari on yksinkertainen: ohjelman käynnistyessä main-metodi aloittaa toimintansa ja ohjelma päättyy, kun sama metodi päättyy. (Wikla, 2005a.) Android-sovelluksista ei siis löydy edellä mainittua main-metodia vaan käytössä on erilaisia herätteitä vaativia metodeja. Esimerkiksi, kun sovellus käynnistetään, etsitään manifestissa määritelty ensimmäisenä käynnistyvä aktiviteetti ja siihen määritelty onCreate-metodi. Nimensä mukaisesti se käynnistyy aina, kun aktiviteetti luodaan. Elinkaaren lopusta löytyy onDestroy ja ennen sitä on yleensä käyty läpi onResume sekä onPause. (Allen, 2012, s. 55.)

Kuten jo aiemmin havaitsimme, Android-laitteiden resurssit ovat rajalliset. Toisin kuin tämän päivän pöytätietokoneissa Android-laitteiden keskusmuisti on hyvinkin rajallinen ja se saattaa täyttyä. Tässä vaiheessa järjestelmän täytyy arvottaa aktiviteettejä ja mahdollisesti tuhota vähemmän tärkeitä aktiviteetteja tärkeämpien tieltä. Esimerkiksi sallimme jonkin pelin sammua taustalta, jos meille tulee puhelu. Tätä varten Androidiin kehitettiin elinkaarimalli. Perinteisessä Javassa ei tämänkaltaisesta elinkaariajattelusta tarvitse välittää. (Allen, 2012, s. 193.)

Kuviossa 3 on kuvattu aktiviteetin elinkaari kokonaisuudessaan.



KUVIO 3 Aktiviteetin elinkaari (Gargenta, 2011, s. 29.)

5.3 Graafiikkakirjastoista

Java-ohjelmoijalle Swing ja AWT ovat tuttuja kirjastoja kaksiulotteisten käyttöliittymien suunnitteluun. Kevyempi Swing-kirjasto on ollut mukana Javan versiosta 1.2 lähtien, kun taas raskaammista komponenteista koostuva AWT julkaistiin 1.0-versiossa. (Wang & Wu, 2009.) Näistä kumpaakaan kirjastoa ei ole käytössä Androidissa, joten Java-ohjelmoijan täytyy totutella hieman toisenlaiseen käyttöliittymäsuunnitteluun.

Swing on kokoelma työkaluja ikkunointiin ja niiden hallintaan. Swingillä voidaan rakentaa yksinkertaisia käyttöliittymiä, jotka koostuvat erilaisista elementeistä, kuten painikkeista ja tekstikentistä. Swing rakennettiin AWT:n päälle ja se tarjoaa monia pitkälle kehitettyjä ominaisuuksia. (Friesen, 2010, s. 407 – 408.) Swingin luokkakirjastot tarjosivat julkaisuvaiheessa melkein 250 luokkaa ja 80 liittymää. Swing ei korvaa AWT:tä, vaikka se rakentuukin AWT:n ydinkirjastojen päälle. Swing kuuluu Javan perustusluokkiin (Java Foundation Classes) yhdessä AWT:n, 2D API:n ja muutaman muun kirjaston kanssa. (Loy, Eckstein, Wood, Elliot & Cole, 2003, s. 1 – 3.)

Vaikka AWT- tai Swing-kirjastoja ei olekaan Androidissa käytettävissä, ei eroavaisuuksia pääpiirteissä juuri ole. Android käsittää kaksi tapaa luoda käyttöliittymiä: XML-pohjaisen komponenttien asettelun tai täysin Java-pohjaisen esittelytavan. Paras tapa on kuitenkin käyttää molempien parhaita puolia. XML-puolella luodaan ja esitellään komponentit sekä asetellaan ne halutulla tavalla, jonka jälkeen Java-kielellä määritellään samojen komponenttien käyttäytyminen. Toisin sanoen XML:llä sanotaan, miltä jokin näyttää ja Javalla, mitä se tekee. Tosin lopulta XML-osuuskin sisällytetään Javan muistiavaruuteen ja Java-koodi on ainoa merkitsevä kieli. (Gargenta, 2011, s. 48.)

6 YHTEENVETO JA POHDINTA

Tässä tutkielmassa tutkittiin Android-sovelluskehityksen erityispiirteitä verrattuna normaaliin pöytätietokoneilla suoritettavaan Java-ohjelmointiin. Aluksi esiteltiin Android-ohjelmoinnin pääpiirteitä ja tyypillinen Hello World –tyyppinen sovellus toteutettuna Androidin versiolla Java-kielestä sekä vertailun vuoksi esiteltiin perinteisen Java-kielen vastine samalle sovellukselle. Seuraavaksi jaoimme erityispiirteet kahteen osaan: laitetasoon sekä ohjelmistotasoon. Laitetasolla tutustuimme mobiililaitteiden asettamiin rajoituksiin sekä esiteltiin Androidin tarjoamia ratkaisuja kyseisille ongelmille. Ohjelmistotason erityispiirteinä havaitsimme Android-käyttöjärjestelmän rakenteen ja Dalvik-virtuaalikoneen määrittämän Javasta poikkeavan ohjelmakoodin kääntämisrakenteen. Lopuksi perehdyimme Java-ohjelmointiin ja siihen miten, sovelluksen elinkaaret eroavat toisistaan sekä esitimme Javan kokoelmien välisiä eroavaisuuksia.

Tutkimuksen edetessä havaittiin Android-ohjelmoinnin ja perinteisen Javan olevan pääperiaatteiltaan hyvin samankaltaisia keskenään. Suurimmat eroavaisuudet havaittiin sovelluksen elinkaaressa sekä Javan luokkakokoelmien erilaisuudessa. Sovelluksen elinkaaren erilaisuus vaatii Android-sovelluksen kehittäjältä hieman erilaista ajattelutapaa, sillä tuttu pääohjelma-ajattelutapa ei enää päde. On pystyttävä käymään läpi sovelluksen erilaisia vaiheita sen syntymästä kuolemaan asti, mutta myös huomioimaan niiden väliin jääneiden elinkaaren vaiheiden vaatimat toiminnallisuudet. Mitä täytyy ottaa huomioon, kun:

- Käyttäjä sulkee sovelluksen?
- Sovellukseni tuhotaan, koska muistinhallinta tarvitsee lisää resursseja?
- Jokin muu sovellus lähettää tietynlaisen viestisignaalin?

Laitetason erityispiirteet ovat käyttäjän kannalta hyvinkin konkreettisia ja niihin pystytään vaikuttamaan laitteen hankintavaiheessa. Sovelluskehittäjän kannalta laitetason vaatimukset ovat toki rajoittavia tekijöitä, mutta samalla

lailla jokaisella muullakin laitealustalla on omat rajansa, joiden sisällä täytyy pystyä toimimaan. Puhuttaessa älypuhelinmaailmasta on selvää, ettei yksinkertaisesti ole mahdollista toteuttaa pöytätietokoneille suunniteltuja raskaita ohjelmistoja.

Ohjelmistotason eroavaisuutena esitetyt järjestelmäpinoon liittyvät tekijät sekä virtuaalikoneen toimintaperiaatteen poikkeavuudet ovat tavalliseen Java-kehitykseen verrattaessa erilaistavia tekijöitä, mutta erilaisuudet piilevät kuitenkin järjestelmän sisemmissä osissa, jolloin sovelluskehitystä tekevän henkilön ei niistä normaalitilanteessa tarvitse suuremmin osin välittää.

Androidin epästandardi Java SE –luokkakirjasto aiheuttaa joitakin eroavaisuuksia sovelluskehityksessä perinteiseen Java-ohjelmointiin verrattuna. Tuttujen grafiikkakirjastojen puute johtaa uudenlaisen käyttöliittymäsuunnittelun opetteluun, mutta pääpiirteiden pysyessä samana ei siitäkään koidu merkittävää eroa. Androidissa grafiikkaa käsitellään XML:n ja Java-kielen sekoituksena, jonka hallitseminen nykyisillä työkaluilla on yksinkertaista. Androidin luokkakirjasto pitää sisällään kuitenkin merkittävän osan tavallisesta Java SE –luokkakirjastosta, jolloin esimerkiksi tietokantojen- ja poikkeuksienhallinta, tekstinkäsittely ja syötteidenhallinta on Java-ohjelmoijalle tuttua.

Android- ja Java-maailma eivät siis eroa toisistaan niin merkittävästi, että erityispiirteitä voisi pitää esteenä Androidiin siirtymisessä. Android tarjoaa ennemminkin uusia mahdollisuuksia ja täysin uudenlaisen maailman, jossa käyttöjärjestelmäversiot kehittyvät nopeaa tahtia ja yhteisö on suuri.

LÄHTEET

Allen, G. (2012). Beginning Android 4. New York: Apress.

Android Developers (2012a, 14. helmikuuta). Designing for Performance. Haettu 18.2.2012 osoitteesta http://developer.android.com/guide/practices/design/performance.html

Android Developers (2012b, 1. helmikuuta). Platform Versions. Haettu 18.2.2012 osoitteesta http://developer.android.com/resources/dashboard/platform-versions.html

Burnette, E. (2008). Java vs. Android APIs. Haettu 8.3.2012 osoitteesta http://www.zdnet.com/blog/burnette/java-vs-android-apis/504

Friesen, J. (2010). Learn Java for Android Development. New York: Apress.

Gargenta, M. (2011). Learning Android. Sebastopol, CA: O'Reilly Media.

Gavalas, D. & Economou, D. (2011). Development Platforms for Mobile Applications: Status and Trends. IEEE Software 28 (1), 77–86.

Goadrich, M., H. & Rogers, M., P. (2011). Smart smartphone development: iOS versus android. Teoksessa M. Rogers P., T. Cortina J., E. Walker L., L. King Smith & D. Musicant R. (toim.) Computer science education: Proceedings of the 42nd ACM technical symposium, (SIGCSE '11). ACM.

Loy, M., Eckstein, R., Wood, D., Elliott, J. & Cole, B. (2003). Java Swing. Sebastopol, CA: O'Reilly Media.

Meier, R. (2010). Professional Android 2 Application Development. Indianapolis, IN: Wiley Publishing Inc.

Paul, Ryan. (2009, 23. helmikuuta). Dream(sheep++): A developer's introduction to Google Android. Haettu 28.3.2012 osoitteesta http://arstechnica.com/open-source/reviews/2009/02/an-introduction-to-google-android-for-developers.ars

Shanker, A. & Lal, S. (2011). Android porting concepts. Teoksessa International Conference on Electronics Computer Technology (ICECT). (s. 129–133).

Sonera. (2012, 16. helmikuuta). Kuuluvuuskartta. Haettu 18.2.2012 osoitteesta http://www.sonera.fi/asiakastuki+ja+edut/puhelin+ja+liittymat/kuuluvuus+ja+nopeuskartta/kuuluvuuskartta

Speckmann, B. (2008). The Android Mobile Platform. Tietotekniikan pro gradu –tutkielma. Itä-Michiganin yliopisto.

TIOBE. (2012, helmikuuta). TIOBE Programming Community Index for February 2012. Haettu 17.2.2012 osoitteesta http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Wang, Y., & Wu, I. (2009). Achieving high and consistent rendering performance of Java AWT/Swing on multiple platforms. Software-practise & experience, 39(7), 701-736

Wikla, A. (2005a). Johdatus ohjelmointiin. Haettu 29.2.2012 osoitteesta http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/3/Metoja.html

Wikla, A. (2005b). Java-kielestä. Haettu 29.2.2012 osoitteesta http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/1/Javat.html