Applikationsstrukturen i Android och iOS

Sebastian Lövdahl, 32485
Kandidatavhandling i datateknik
Åbo Akademi
Institutionen för informationsteknologi
Handledare: Linda Mannila
2011

Innehållsförteckning

Referat

De krav som ställs på applikationer för mobila enheter skiljer sig från de krav som ställs på applikationer utvecklade för vanliga datorer. Begränsningar finns på till exempel mängden ledigt minne och skärmutrymmet.

Idag finns det många olika utvecklingsplattformar för mobila enheter. Till de mest populära hör Android, iOS, Symbian, BlackBerry OS och Windows Phone. Plattformarna skiljer sig från varandra på ett flertal olika sätt, till exempel på vilket programmeringsspråk som används för applikationsutveckling och hur applikationer exekveras i operativsystemet. Andra saker som skiljer sig är utvecklingsmiljöer och hur distributionen av applikationer från utvecklare till användare görs.

I den här avhandlingen jämförs applikationsstrukturerna i Android och iOS med varandra.

Nyckelord: applikationsstruktur, mobila utvecklingsplattformar, Android, iOS

1 Introduktion

1.1 Smarttelefonernas uppkomst

Mycket har hänt i mobiltelefonibranschen sedan de första experimenten med mobil telefonteknologi gjordes i USA på 1920-talet [THE2007]. Mobiltelefonerna har blivit mindre och lättare, fått längre batteritid, flera funktioner och större skärmar jämfört med de tidiga mobiltelefonerna från 50-, 60- och 70-talet. I början av 90-talet började handdatorer (eng. *Personal Digital Assistant, PDA*) komma ut på marknaden. Handdatorerna på den tiden hade funktioner som kalender, adressbok, e-postklient, webbläsare och anteckningsblock. Dessutom kunde de flesta handdatorer synkroniseras med en dator, vilket gjorde att de blev populära inom bland annat företagsvärlden. Utvecklingen av smarttelefoner (eng. *smartphones*) kom igång i mitten av 90-talet [PEI2006].

En smarttelefon skiljer sig från andra mobiltelefoner genom att ha många andra funktioner än telefon och SMS. Förutom de funktioner som handdatorer har är vanliga kännetecken för en modern smarttelefon relativt snabb processor, relativt mycket minne, fysiskt eller virtuellt tangentbord, pekskärm, mediaspelare, kamera och möjlighet att installera applikationer [HAR2009]. Försäljningen av smarttelefoner har nu kommit i gång på allvar, mellan slutet av 2009 och slutet av 2010 ökade försäljningen av smarttelefoner med 88,6 % [GOG2011].

1.2 Mobila operativsystem

Ett operativsystem kan allmänt definieras som den mjukvara som utgör gränssnittet mellan hårdvara och användare och som möjliggör exekvering av annan mjukvara. Operativsystemet ger applikationsutvecklare ett standardiserat sätt att interagera med hårdvara. Till de vanligaste och mest allmänna operativsystemen för stationära datorer hör Windows, Linux och Mac OS X. Operativsystem för vanliga datorer lämpar sig ändå inte för att användas direkt på mobila enheter, de är för stora och resurskrävande [AND2009].

Sedan smarttelefonernas intåg på marknaden har ett flertal mobila plattformar

utvecklats. Till de mest populära hör *Symbian OS*, *BlackBerry OS*, *Palm OS*, *Android*, *iOS*, *Windows Mobile*, *MeeGo* och *BREW*. En mobil utvecklingsplattform innehåller förutom operativsystem även mjukvaruutvecklingsverktyg och bibliotek och ramverk för utveckling mot plattformen [PEI2006]. De plattformar som är överlägset mest populära för tillfället är Android, iOS och BlackBerry OS. Tillsammans såldes över 20 miljoner mobiltelefoner med de här tre operativsystemen tredje kvartalet 2010, medan resten av operativsystemen delade på ca 1 miljoner sålda enheter [MAR2011].

Android och iOS hade under tredje kvartalet 2010 marknadsandelar på 39 % respektive 27 % vardera medan BlackBerry OS hade 28 % av marknaden. Därefter kom Symbian OS med en marknadsandel på 3 % [MAR2011].

1.3 Avhandlingens begränsning och struktur

I den här avhandlingen har jag valt att göra en jämförelse mellan två mobila operativsystem, Android och iOS, eftersom de för tillfället är de mest populära och intressanta plattformarna.

Syftet med avhandlingen är att utreda skillnaderna mellan applikationsstrukturen i Android och iOS. Efter den korta introduktionen till mobila operativsystem som givits i detta kapitel presenteras de båda plattformarna som avhandlingen inriktar sig på noggrannare i kapitel två. I kapitel tre gås de olika aspekterna av applikationsstrukturen igenom och en exempelapplikation per plattform presenteras. I den avslutande diskussionen jämförs plattformarna med varandra och reflektioner över jämförelsen upp.

2 Presentation av Android och iOS

I det här kapitlet kommer plattformarna Android och iOS att gås igenom mera ingående med bland annat en kort överblick över fakta om de båda plattformarna.

2.1 Android

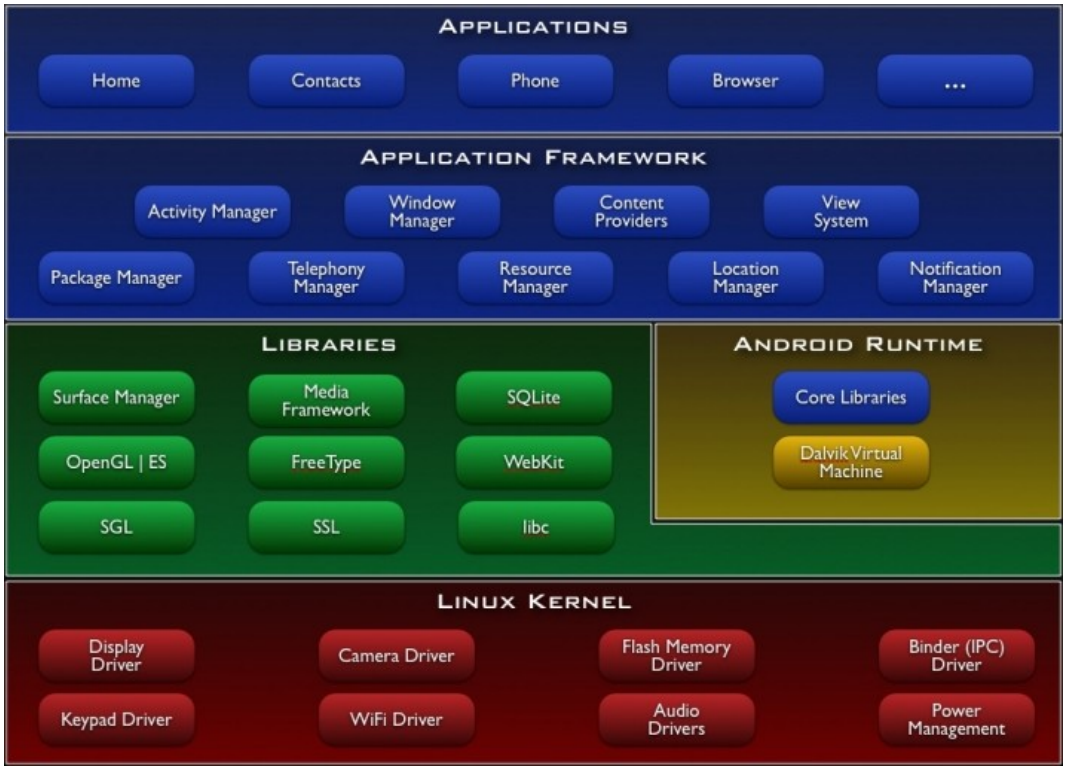
Android är ett mjukvarupaket som består av operativsystem, mellanvara (eng. *middleware*) och applikationer utvecklat av *Open Handset Alliance* (OHA) främst för mobila enheter. OHA är en företagssammanslutning som grundades under ledning av Google i november 2007 och som idag består av 80 företag i mobiltelefoni- och teknologibranschen. OHA:s mål är att utveckla och upprätthålla öppna standarder för mobila enheter. Android offentliggjordes samtidigt som OHA grundades och är OHA:s första och enda produkt än så länge [OHA2011]. Knappt ett år senare, i september 2008, var version 1.0 av Android klar [ANN2008]. En månad senare kom den första kommersiellt tillgängliga telefonen ut på marknaden, T-Mobile G1 tillverkad av den taiwanesiska mobiltelefontillverkaren HTC [HTC2008]. I mitten av 2010 fanns det över 40 Android-telefoner från mera än 10 olika tillverkare ute på marknaden [JAM2011].

Källkoden till Android finns fritt tillgänglig på internet licensierad under den fria Apache-licensen vilket innebär att i princip vem som helst kan bidra med förbättringar och nya funktioner till källkoden. Det är också fritt fram för vilken tillverkare och privatperson som helst att använda och sälja produkter med Android på. Android kan köras på många olika sorters enheter och inbyggda system även om mobiltelefonerna och surfplattorna är de produkter på vilka Android är mest populär för tillfället. Exempel på andra enheter som Android finns på är netbooks, TV-apparater, digitala TV-mottagare och i bilar [JAM2011].

Efter version 1.0 av Android har utvecklingen av plattformen fortsatt i snabb takt. I april 2009 kom version 1.5, i september samma år kom version 1.6, i januari 2010 kom version 2.1, i maj 2010 kom version 2.2 och i december 2010 kom

version 2.3. Android är under konstant utveckling och många nya funktioner tillkommer i varje ny version [AND2011].

Med Android avses en helhetslösning som inkluderar alla nödvändiga delar för mobila enheter och applikationsutveckling för de mobila enheterna. Först och främst består Android av ett operativsystemet och alla bibliotek och applikationer som körs på den mobila enheten. Utöver det innehåller Android en uppsättning av mjukvaruutvecklingsverktyg (eng. *Software Development Kit, SDK*) och nativa utvecklingsverktyg (eng. *Native Development Kit, NDK*) som stöd för utveckling av applikationer specifika för Android-operativsystemet [AND2011].



Figur 1: Schema över Android-operativsystemets interna struktur

Grunden i Android-operativsystemet består av en modifierad version av den fritt tillgängliga *Linux*-kärnan. Linux-kärnan sköter om kommunikationen med hårdvaran i den mobila enheten (se figur 1). Ovanpå Linux-kärnan finns abstrakta bibliotek skrivna i C och C++ för bland annat 2D-grafik, accelererad 3D-grafik,

säkerhet och kryptering, musikuppspelning, videouppspelning, röstinspelning, webbvisning och strukturerad datalagring. Vid sidan av biblioteken finns Androids *virtuella maskin Dalvik* (eng. *Dalvik Virtual Machine*) som sköter om exekveringen av alla applikationer [JAM2011].

Android-SDK:n innehåller bland annat verktyg för avlusning (eng. *debugging*) och design av användargränssnitt samt en emulator som emulerar en Android-telefon. I SDK:n finns också en insticksmodul (eng. *plugin*) till utvecklingsmiljön *Eclipse* som lägger till funktioner i Eclipse som underlättar utvecklingen av applikationer för Android. Eclipse är en utvecklingsmiljö baserad på öppen källkod som används främst vid utveckling i programmeringsspråken Java- och C++. Det går att utveckla Android-applikationer med vilken texteditor som helst, men Eclipse rekommenderas [AND2011].

Programmeringsspråket som används för utveckling av Android-applikationer är *Java*. Det är ändå inte samma samma sak som den Java-plattform som utvecklats av företaget *Sun*, applikationer exekveras till exempel i den virtuella maskinen Dalvik istället för i *Java VM* (eng. *Java Virtual Machine*). Utvecklare har ändå tillgång till några få av de standardbibliotek som hör till Java-plattformen [ERI2008]. Utöver det finns det Android-specifika bibliotek som ersätter en stor del av Java-biblioteken och dessutom ger tillgång till saker som till exempel telefonfunktioner, bluetooth, platstjänster och funktioner för ett användargränssnitt anpassat för mobila enheter. Det är också möjligt att utveckla applikationer i programmeringsspråken C och C++ med hjälp av Android NDK. Men eftersom det gör applikationerna mera komplexa rekommenderas det endast till exempel för tunga beräkningar eller vid återanvändning av kod [AND2011].

2.2 iOS

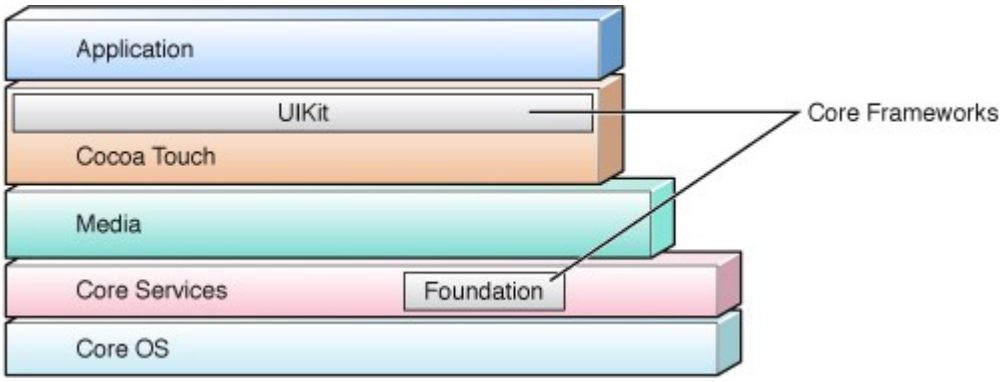
iOS är en helhetslösning av mjukvara för mobila enheter utvecklat av *Apple*. Källkoden är inte fritt tillgänglig och iOS finns endast tillgänglig på Apples egna produkter *iPhone*, *iPad*, *iPod Touch* och *Apple TV*. Apple släppte den första iPhonen i juni 2007, men då kallades operativsystemet endast för *OS X för*

iPhone. Efter det har namnet ändrats två gånger: först i mars 2008 till *iPhone OS* och därefter i juni 2010 till iOS [CHR2011].

För version 1.0 av iOS fanns ingen möjlighet att utveckla eller installera nativa applikationer. Med version 2.0 som släpptes i juli 2008 tillkom möjligheten att installera nativa applikationer och en iOS-SDK (*Software Development Kit*) gjordes tillgänglig. I den efterföljande versionen 3.0 som släpptes i juni 2009 fanns inga lika revolutionerande nyheter. Version 4.0 som släpptes i juni 2010 introducerade möjlighet till riktig *multitasking* (sv. parallellbearbetning) för alla applikationer, vilket i tidigare versioner bara var möjligt för Apples egna applikationer [CHR2011].

Applikationer för iOS utvecklas i programmeringsspråket *Objective-C*. Det är en vidareutveckling av programmeringsspråket C med stöd för objektorienterad programmering. *Xcode* är en utvecklingsmiljö gjord av Apple specifikt för att utveckla applikationer till Mac OS X som från och med version 4 är avgiftsbelagd. I iOS-SDK:n finns en insticksmodul till Xcode som ger stöd för utveckling för iOS. Förutom det innehåller SDK:n också en iPhone-, iPod- och iPad-emulator [IOSD2011].

iOS-operativsystemet är en vidareutveckling av *Mac OS X*. iOS körs på samma kärna som OS X gör, dvs. en *Unix*-baserad kärna med namnet *Darwin*. iOS är en vidareutveckling som är anpassad för mobila enheter, med stöd för bland annat pekskärmar och andra mobilrelaterade funktioner. I figur 2 illustreras den interna strukturen i iOS med den lägsta abstraktionsnivån *Core OS* längst ner, den högsta abstraktionsnivån *Cocoa Touch* högst upp och ovanpå den alla applikationer. Till *Core OS* hör *Darwin*-kärnan, filsystemet, nätverksstacken, strömhantering andra lågnivådrivrutiner. På *Core Services*-nivån finns gränssnitt på en högre abstraktionsnivå mot hårdvara som till exempel GPS, kompass och accelerometer. *Media*-nivån består av gränssnitt för till exempel ljud- och videouppspelning medan *Cocoa Touch* innehåller alla ramverk för användargränssnittet [IOSD2011]. Ramverken presenteras mera ingående i kapitel 3.2.1.



Figur 2: Strukturen i iOS-operativsystemet

3 Applikationsstrukturen i Android och iOS

3.1 Android

3.1.1 Start av applikation

Android-applikationer har alltid en huvudingångsstig. Men en `main()`-funktion som körs varje gång applikationen startas saknas, istället definieras en klass som huvudklass. Det kan även finnas andra ingångsstigar i applikationen som kan användas i olika specialfall, applikationer kan till exempel anropa en specifik del av andra applikationer för att utföra en uppgift, och det behöver inte vara samma klass som huvudingångsstigen. I Android rekommenderas det att befintliga komponenter återanvänds så ofta som möjligt och utvecklare rekommenderas i mån av möjlighet programmera egna komponenter så att andra applikationer kan återanvända dem [JAM2011].

Varje applikation i Android körs som en unik användare i en egen instans av den virtuella maskinen Dalvik. Det innebär att alla applikationer är isolerade från varandra och inte kan läsa från eller skriva till andra applikationers filer. Dalvik är optimerad för att kunna köra flera parallella instanser av sig själv effektivt [AND2011].

En applikation för Android distribueras i en arkivfil med filändelsen *.apk*. Arkivet

innehåller alla filer som hör till en specifik applikation: exekverbar fil, bilder, ljudfiler, konfigurationsfiler och andra datafiler. För att en applikation överhuvudtaget ska kunna startas måste en fil med namnet *AndroidManifest.xml* finnas med. Den definierar ett antal olika nödvändiga för applikationen, till exempel ett unikt Java-paketnamn, applikationens huvudingångsstig, vilka olika komponenter och huvudklasser som finns i applikationen, vilka behörigheter applikationen behöver, vilken version av Android som krävs och vilka bibliotek applikationen ska länkas med. Den exekverbara filen är en fil i *.dex*-format (eng. *Dalvik Executable*) som innehåller klasser kompilerade av en Java-kompilator och som är optimerade för minimal minnesanvändning [AND2011].

3.1.2 Grundläggande beståndsdelar

En Android-applikation består av åtminstone en men oftast flera eller alla av kärnklasserna som listas i tabell 1. En *aktivitet* hanterar och visar en specifik uppgift på skärmen i en applikation, till exempel kan en kalenderapplikation innehålla en aktivitet för att visa månadsvyn och en annan aktivitet för att lägga till nya händelser i kalendern. En *tjänst* i sin tur körs i bakgrunden och har inget direkt användargränssnitt och kan köras så länge eller ofta som programmeraren vill. Ett exempel på en tjänst är en musikuppspelningsapplikation, där själva musikuppspelningen är en tjänst som körs i bakgrunden och användargränssnittet som visar vilken låt som spelas och möjliggör byte av låt är en aktivitet. Användaren kan avsluta låtgränssnittet och göra någonting annat, musikuppspelningen fortsätter ändå köras i bakgrunden så att musiken inte slutar spelas [JAM2011].

Tjänsteleverantörer (eng. *content providers*) möjliggör lagring av information som kan delas mellan alla applikationer. Ett bra exempel på en tjänsteleverantör som finns färdigt i Android är `ContactsContract` som kan lagra uppgifter om personer som namn, telefonnummer, adress och så vidare. Meddelandemottagare (eng. *broadcast receivers*) i sin tur tar emot meddelanden från operativsystemet och andra applikationer och vidtar eventuella åtgärder. Exempel på meddelanden från operativsystemet är låg batterinivå, kameraknappen trycktes in eller en ny applikation installerades [AND2011].

Namn	Översättning	Basklass i `android`
Activity	aktivitet	`app.Activity`
Service	tjänst	`app.Service`
Content provider	tjänsteleverantör	`content.ContentProvider`
Broadcast receiver	meddelandemottagare	`content.BroadcastReceiver`

Tabell 1: Kärnklasserna i Androids ramverk.

Intents är ett ramverk för asynkron kommunikation mellan operativsystemet och aktiviteter, mellan operativsystemet och tjänster, mellan aktiviteter och tjänster samt inom aktiviteter och tjänster. *Intents* används främst för att skicka en abstrakt beskrivning på vad som ska göras eller någonting viktigt som har hänt i systemet. Aktiviteter, tjänster och sändningsmottagare aktiveras med hjälp av *intents*. I musikuppspelningsexemplet skulle *intents* användas i aktiviteten för låtgränssnittet för att berätta åt uppspelningstjänsten om till exempel en ny låt ska spelas upp eller om uppspelningen ska pausas. Om uppspelningstjänsten inte är igång startas den då *intenten* skickas från låtgränssnittet [JAM2011].

Intent-anrop kan göras implicit eller explicit. *Explicit* betyder att en klass anropas direkt med namnet på aktiviteten eller klassen medan *implicit* betyder att ett allmänt anrop som till exempel `ACTION_DIAL` görs och operativsystemet avgör vilken aktivitet eller klass som ska anropas. Klasser kan definiera *intent-filter*, vilket innebär att operativsystemet vet vilka typer av explicita *intent*s klassen kan ta emot [AND2011].

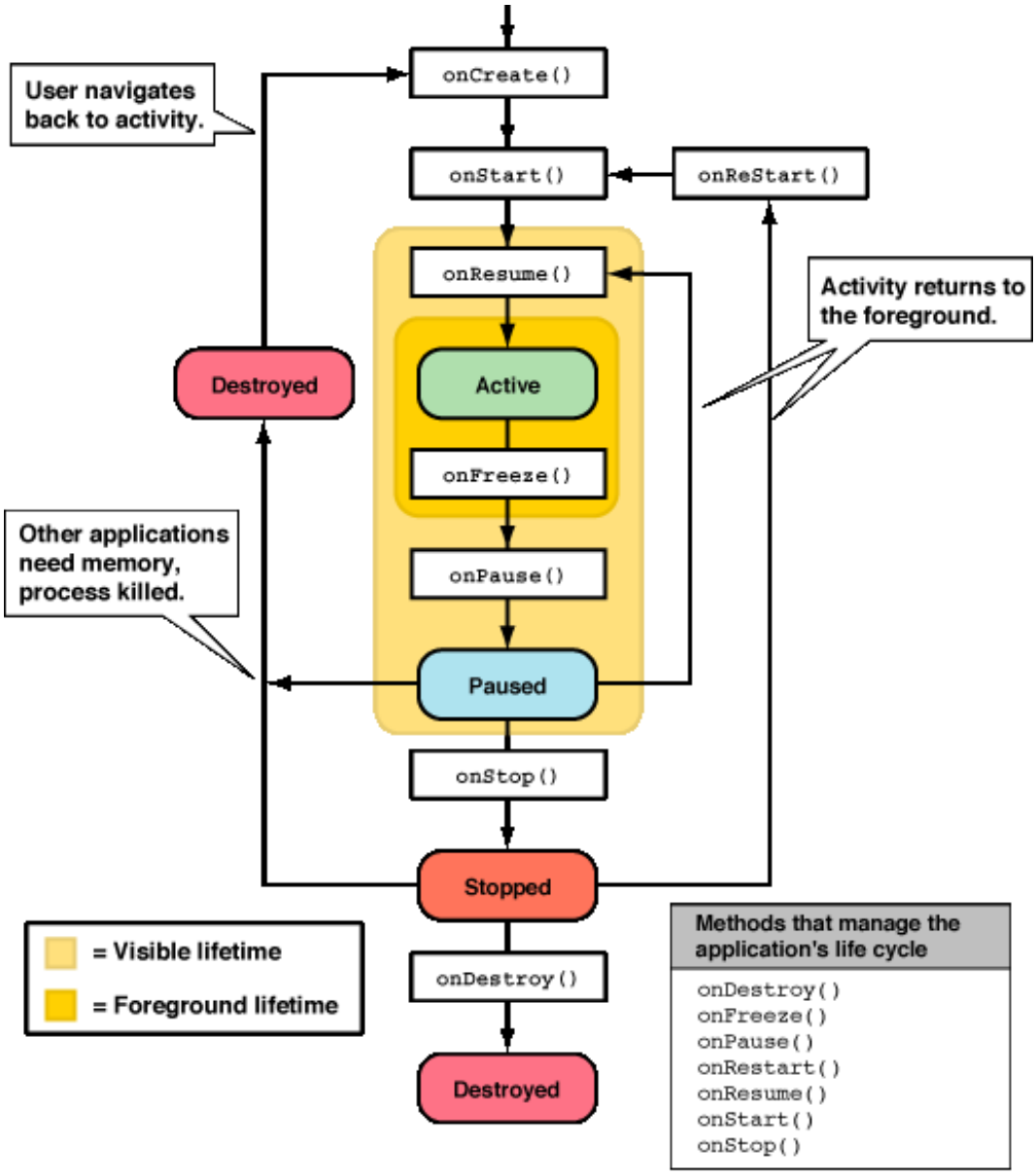
Som grund för nästan allting som ritas upp på skärmen finns klassen `android.view.View`. Varje `View` ansvarar för ett rektangulärt område av skärmen och för alla händelser i det området. En typ av objekt som `View` är basklass för är så kallade *widgets*, dvs. knappar, textrutor, kryssrutor och så vidare. `ViewGroup` är en annan typ av `View`-objekt som används för att gruppera andra `Views` på skärmen [JAM2011].

3.1.3 Applikationens livscykel

Varje aktivitet (`android.app.Activity`) i en applikation har sin egen livscykel som följer grafen i figur 3. En aktivitet som är startad och finns i minnet kan befinna sig i tre olika tillstånd: aktiv (eng. *running*), pausad (eng. *paused*) eller stoppad (eng. *stopped*). En aktivitet skapas genom att anropa dess `onCreate()`-funktion och den skapas då i stoppad-läget. I `onCreate()` måste aktivitetens användargränssnitt definieras och initialiseras. Nästa funktion som anropas är `onStart()` och aktiviteten är då pausad men synlig för användaren, ändå inte nödvändigtvis längst fram på skärmen. Funktionen `onResume()` anropas då aktiviteten ska få fokus och visas längst fram på skärmen och aktiviteten är då i aktiv-läget [AND2011].

Då någon annan aktivitet får fokus och visas längst fram på skärmen anropas först `onPause()`-funktionen för att berätta åt aktiviteten att den inte körs i förgrunden längre och den sätts då i läget pausad. Den gamla aktiviteten kan ändå i vissa fall visas i bakgrunden bakom den nya aktiviteten. Den nya aktivitetens `onCreate()` anropas inte före `onPause()` för den föregående aktiviteten har körts färdigt, `onPause()`-funktionen måste alltså köras snabbt. I `onPause()` ska alla tunga beräkningar avslutas och alla viktig data sparas, eftersom risken finns för att systemet dödar aktiviteten om mera minne behövs [AND2011].

Efter `onPause()` kommer ofta ett anrop till `onStop()` direkt efteråt. Det garanteras inte att `onStop()` alltid körs före en aktivitet avslutas, alla data som måste sparas på disk ska sparas redan i `onPause()`. Ett anrop till `onStop()` meddelar aktiviteten att den inte längre visas överhuvudtaget och den ligger i bakgrunden i stoppad-läget och väntar på antingen `onRestart()` eller `onDestroy()`. Före aktiviteten avslutas helt och hållet, dvs. tas bort ur minnet, anropas `onDestroy()`-funktionen [AND2011].



Figur 3: Schema över en aktivitets livscykel i Android

3.2 iOS

3.2.1 Start av applikation

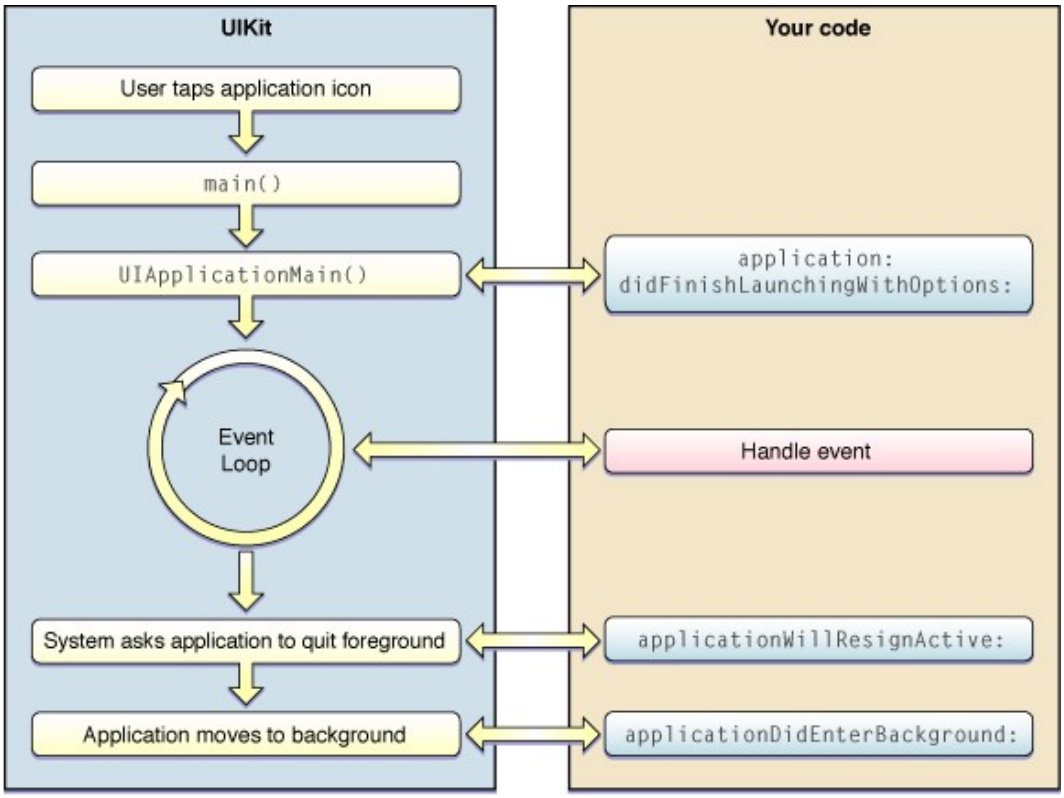
iOS-applikationer har endast en ingångsstig. Som i alla andra C-liknande programmeringsspråk finns det i Objective-C en `main()`-funktion som exekveras då applikationen startas. En applikation består inte av flera olika

komponenter som kan startas oberoende av varandra. Det finns fem olika tillstånd som en applikation kan befinna sig i: avslutad (eng. *not running*), inaktiv (eng. *inactive*), aktiv (eng. *active*), bakgrund (eng. *background*) och avbruten (eng. *suspended*). Vid övergången mellan de här tillstånden anropas följande delegerade funktioner i `UIApplicationMain`-klassen:

- `application:didFinishLaunchingWithOptions:`
- `applicationDidBecomeActive:`
- `applicationWillResignActive:`
- `applicationDidEnterBackground:`
- `applicationWillEnterForeground:`
- `applicationWillTerminate:`

Före version 4.0 av iOS var en applikations livscykel väldigt enkel. Applikationen startades, kördes så länge användaren ville, avslutades och togs bort ur minnet senast fem sekunder efter att den avslutats. Från och med version 4.0 har *multitask*-möjlighet tillkommit och applikationer kan nu fortsätta köras i bakgrunden även om deras användargränssnitt döljs. I figur 4 beskrivs en förenklad modell av en applikations livscykel i iOS 4.0 där *multitask* är möjligt. [IOSD2011].

En iOS-applikation distribueras i en arkivfil med filändelsen *.ipa* [IOSD2011]. Arkivet innehåller alla filer som hör till en specifik applikation: en exekverbar applikationsfil, ikoner, bilder, ljudfiler, konfigurationsfiler och andra datafiler. En obligatorisk fil förutom den exekverbara filen är *Info.plist* som beskriver applikationen för operativsystemet. Den ska till exempel innehålla namnet på den exekverbara filen, en unik identifierare som operativsystemet använder och det namn på applikationen som visas åt användaren. En viktig typ av filer som också finns i arkivet är så kallade *nib*-filer med filändelsen *.xib*. *Nib*-filer innehåller adresserbara beskrivningar av användargränssnitt i XML-format [ERI2008].



Figur 4: Schema över en applikations livscykel i iOS

3.2.2 Grundläggande beståndsdelar

Utvecklingen av applikationer för iOS påminner mycket om utvecklingen av applikationer för Mac OS X. Många av ramverken och gränssnitten som finns i OS X finns också i iOS, bland annat *Core Services* och *Media*. Dessutom finns en vidareutveckling av *Cocoa*-biblioteket från OS X i iOS med namnet *Cocoa Touch*, som är ett av de viktigaste och mest använda biblioteken för utveckling av iOS-applikationer. *Core Services* är ett ramverk med funktioner för saker som till exempel samlingar (eng. *Collections*), stränghantering, platstjänster (eng. *location awareness*), SQLite-databaser, adressbok och nätverkskommunikation. Media i sin tur är ramverket för bildvisning, animationer, ljud och videouppspelning medan *Cocoa Touch*-ramverket innehåller grundläggande funktioner som alla applikationer använder sig av [BIL2009]. De viktigaste ramverken inom *Cocoa Touch*-biblioteket är `Foundation` och `UIKit`. `Foundation`-ramverket innehåller en mängd olika klasser som används i alla applikationer, till exempel klasser för strängar, tal och andra datatyper, *Collection*-klasser så som länkade listor, hashtabeller och arrayer samt klasser för hantering av tid och tidzoner.

`UIKit` i sin tur består av bland annat klasser för användargränssnitt, händelsehantering och applikationers livscykel [BIL2009].

3.2.2 Designmönster

Programmeringsramverken i iOS är uppbyggda enligt olika designmönster (eng. *design patterns*). En viktig del i att lära sig att utveckla applikationer för iOS är att lära sig hur de olika designmönstren fungerar och används. De viktigaste designmönstren är *objektorienterad programmering* och *Model-View-Controller*-mönstret (*MVC*-mönstret) [ERI2008]. Även delegering (eng. *delegation*) är ett viktigt designmönster [DAV2009].

MVC-mönstret delar upp all funktionalitet i tre separata och distinkta kategorier. *Model* innehåller alla klasser som lagrar, hanterar och utför beräkningar på själva datan i applikationen medan *View* består av själva användargränssnittet med knappar, rutor och element som användaren kan interagera med. *Controller* binder ihop *Model* och *View* genom att reagera på användarens input till *View*-klasserna och föra informationen vidare till rätt *Model*-klass, som i sin tur kan ha *Controller* att uppdatera *View* med nya data. En *View*-klass har alltså ingen aning om vad som ska hända då till exempel en knapptryckning görs och en *Model*-klass vet inte hur och var datan ska visas eller hämtas från. Genom att använda *MVC*-mönstret underlättas återanvändningen av klasser, funktionalitet och användargränssnitt eftersom varje klass då har en väldigt specifik uppgift [DAV2009].

Delegering innebär att man i en klass ger över ansvaret för vissa händelser till ett annat objekt. En klass i `UIKit` som till exempel skapar en tabell innehåller ingen funktionalitet för att bestämma vad som ska hända då användaren pekar på en rad, istället delegeras den till en funktion som utvecklaren har valt [IOSD2011].

I iOS byggs gränssnittet upp av två viktiga basklasser: `UIView` och `UIViewController`. De här två klasserna ansvarar för att definiera och placera alla objekt och element på skärmen. Exempel på klasser som ärver av `UIView` är `UIButton`, `UIDatePicker`, `UITextView` och `UISearchBar` [ERI2008].

3.3 Genomgång av kodexempel

För att illustrera skillnaderna mellan applikationsstrukturen i Android och iOS kommer här ett ”Hello, World!”-exempel för vardera plattformen att presenteras.

3.3.1 Hello World för Android

Android-versionen av ”Hello, World!” innehåller en kodfil, *HelloAndroid.java*, och två resursfiler: *AndroidManifest.xml* och *res/values/strings.xml*.

AndroidManifest.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest
3.   xmlns:android="http://schemas.android.com/apk/res/android"
4.   package="example.helloandroid"
5.   android:versionCode="1"
6.   android:versionName="1.0">
7.
8.   <application
9.     android:icon="@drawable/icon"
10.    android:label="@string/app_name">
11.    <activity android:name=".HelloAndroid"
12.      android:label="@string/app_name">
13.      <intent-filter>
14.        <action android:name="android.intent.action.MAIN" />
15.        <category
16         android:name="android.intent.category.LAUNCHER" />
17.      </intent-filter>
18.    </activity>
19.  </application>
20.</manifest>
```

I resursfilen *AndroidManifest.xml* definieras paketnamnet på rad 4, applikationens ikon och namn på rad 9 och 10 samt aktiviteten `HelloAndroid` på rad 11 till 18. För aktiviteten definieras ett *intent-filter* (rad 13-17) som anger att aktiviteten

är applikationens huvudaktivitet och att den ska aktiveras då applikationen startas. På rad 12 visas ett exempel på hur hänvisningar till andra resursfiler görs då en sträng hämtas från *strings.xml*-filen med `@string/app_name`.

res/values/strings.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.   <string name="app_name">helloandroid</string>
4. </resources>
```

I resursfilen *values/strings.xml* definieras endast namnet på applikationen i strängen `app_name` (rad 3).

HelloAndroid.java:

```
1. package com.example.helloandroid;
2. import android.app.Activity;
3. import android.os.Bundle;
4. import android.widget.TextView;
5. public class HelloAndroid extends Activity {
6.    @Override
7.    public void onCreate(Bundle savedInstanceState) {
8.          super.onCreate(savedInstanceState);
9.          TextView tv = new TextView(this);
10.         tv.setText("Hello, World!");
11.         setContentView(tv);
12.   }
13. }
```

På första raden i kodfilen (rad 1) deklareras vilket paket som applikationen hör till. Alla kodfiler som hör till samma applikation måste höra till samma paket. Sedan importeras klasserna från Android-ramverket som används i filen (rad 2-4). Därefter börjar den egentliga programkoden.

Först skapas en ny klass med namnet `HelloAndroid` (rad 5) som är en utökning av `Activity`-klassen. Nästa kodrad (rad 6), `@Override`, indikerar åt

Java-kompilatorn att efterföljande funktion överskuggar superklassens funktion med samma namn. För att en applikation ska kunna startas och visa någonting på skärmen måste funktionen `onCreate()` implementeras (rad 7).

Funktionen `onCreate()` tar en parameter av typ `Bundle` som kan innehålla information om aktivitetens tillstånd om den skapas igen efter att ha avslutats. Första raden i funktionskroppen (rad 8) är ett funktionsanrop som måste göras, annars kastas ett undantag och aktiviteten avslutas. Superklassens `onCreate()` anropas, dvs. `Activity`-klassens `onCreate()`. Efter det kan initialiseringen av användargränssnittet påbörjas. Här (rad 9) initialiseras en widget av typen `TextView` för ”Hello, World!”-texten. En `TextView` är en typ av `View` som visar text på skärmen och som kan låta användaren redigera texten. På följande rad (rad 10) anropas funktionen `setText()` i `TextView` med parametern ”Hello, World!” och den anger vilken textsträng som `TextView`en ska visa. Till sist anropas `setContentView(tv)` i `HelloAndroid`-klassen (rad 11) för att ange vilken `View` som aktiviteten ska visa.

3.3.2 Hello World för iOS

iOS-versionen av applikationen innehåller fem filer med kod: *main.m*, *HelloWorldAppDelegate.h*, *HelloWorldAppDelegate.m*, *MyView.h* och *MyView.m*. Applikationen innehåller även två resursfiler: *Info.plist* och *MainWindow.xib*. Filen *MainWindow.xib* är en stor fil som genereras automatiskt av Xcode. Den behöver inte ändras för den här applikationen och därför tas den inte upp här.

Info.plist:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3. <plist version="1.0">
4. <dict>
5.    <key>CFBundleDevelopmentRegion</key>
6.    <string>English</string>
7.    <key>CFBundleDisplayName</key>
8.    <string>${PRODUCT_NAME}</string>
```

```
9.    <key>CFBundleExecutable</key>
10.   <string>${EXECUTABLE_NAME}</string>
11.   <key>CFBundleIconFile</key>
12.   <string></string>
13.   <key>CFBundleIdentifier</key>
14.   <string>com.yourcompany.$
  {PRODUCT_NAME:rfc1034identifier}</string>
15.   <key>CFBundleInfoDictionaryVersion</key>
16.   <string>6.0</string>
17.   <key>CFBundleName</key>
18.   <string>${PRODUCT_NAME}</string>
19.   <key>CFBundlePackageType</key>
20.   <string>APPL</string>
21.   <key>CFBundleSignature</key>
22.   <string>????</string>
23.   <key>CFBundleVersion</key>
24.   <string>1.0</string>
25.   <key>LSRequiresIPhoneOS</key>
26.   <true/>
27.   <key>NSMainNibFile</key>
28.   <string>MainWindow</string>
29. </dict>
30. </plist>
```

I *Info.plist* definieras bland annat den exekverbara filens namn (rad 9-10), en unik sträng som identifierar applikationen i operativsystemet (rad 13-14), applikationsnamnet (rad 17-18), vilken version av applikationen det är (rad 23-24) och vilken *nib*-fil som ska laddas då applikationen startas (rad 27-28).

main.m:

```
1. #import <UIKit/UIKit.h>

2. int main(int argc, char *argv[]) {
3.    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
4.    int retVal = UIApplicationMain(argc, argv, nil, nil);
5.    [pool release];
6.    return retVal;
7. }
```

Filen *main.m* genereras automatiskt av Xcode då ett nytt projekt skapas. På rad 1 importeras biblioteket `UIKit`. Därefter kommer applikationens `main`-funktion (rad 2-7) som körs då applikationen startas. Funktionens uppgift är att initialisera minneshanteringen för applikationen (rad 3) och att starta applikationens huvudslinga (rad 4).

HelloWorldAppDelegate.h:

```
1. #import <UIKit/UIKit.h>

2. @interface HelloWorldAppDelegate : NSObject
  <UIApplicationDelegate> {
3.      UIWindow *window;
4. }

5. @property (nonatomic, retain) IBOutlet UIWindow *window;

6. @end
```

HelloWorldAppDelegate.h implementerar klassen `HelloWorldAppDelegate` som fungerar som *delegate* åt `UIApplication` (rad 2) och innehåller en variabel `UIWindow` (rad 3) som representerar fönstret där `UIView`en ska skrivas. På rad 5 definieras funktioner för att bland annat visa och ändra på `UIWindow`-variablen med `@property`. Utöver det definieras den som en `IBOutlet`, vilket anger att den ska kopplas till ett objekt i *MainWindow.xib*-filen.

HelloWorldAppDelegate.m:

```
1. #import ”HelloWorldAppDelegate.h”
2. #import ”MyView.h”

3. @implementation HelloWorldAppDelegate

4. @synthesize window;

5. - ((BOOL)application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
6.   MyView *view = [[MyView alloc] initWithFrame:[window frame]];
7.   [window addSubview:view];
```

```
8.    [view release];
9.    [window makeKeyAndVisible];
10.   return YES;
11. }

12. - (void)dealloc {
13.    [window release];
14.    [super dealloc];
15. }

16. @end
```

HelloWorldAppDelegate.m implementerar klassen `HelloWorldAppDelegate` (rad 3) och dess funktioner (rad 5-15). Rad 4 anger att `@property`-deklarationen från *HelloWorldAppDelegate.m* ska implementeras. Funktionen `didFinishLaunchingWithOptions` (rad 5) överskuggas här och den anropas då applikationen startas. I funktionen allokeras minne för den `UIView` (rad 6) som ska innehålla textsträngen och därefter läggs den till `UIWindow`-instansen (rad 7). Efter det görs fönstret till huvudfönster och visas längst fram på skärmen (rad 9). Funktionen `dealloc()` anropas då applikationen avslutas och den ser till att allt allokerat minne frigörs (rad 12-15).

MyView.h:

```
1.#import <UIKit/UIKit.h>

2.@interface MyView : UIView {
3.}

4.@end
```

I *MyView.h* definieras klassen `MyView` som en subklass till `UIView` (rad 2).

MyView.m:

```
1. #import "MyView.h"

2. @implementation MyView
```

```
3. - (void)drawRect:(CGrect)rect {
4.     NSString *hello = @”Hello, World!”;
5.     CGPoint location = CGPointMake(10, 20);
6.     UIFont *font = [UIFont systemFontSize:24];
7.     [[UIColor whiteColor] set];
8.     [hello drawAtPoint:location withFont:font];
9. }

10. - (void)dealloc {
11.     [super dealloc];
12. }

13. @end
```

Filen *MyView.m* implementerar klassen `MyView` (rad 2). Funktionen `drawRect()` som överskuggas på rad 3 initialiserar först en sträng med texten *”Hello, World!”* (rad 4). Därefter skapas en `CGPoint` (rad 5) med hänvisning till en punkt i ett tvådimensionellt koordinatsystem där textsträngen ska börja skrivas ut. Sedan initialiseras en variabel som anger fonten och fontstorleken på texten (rad 6). På rad 7 sätts textfärgen och på rad 8 skrivs texten ut på skärmen.

4 Diskussion

4.1 Minneshantering

En fundamental skillnad mellan Android och iOS är de programmeringsspråk som används. För utveckling till Android används Java och för iOS Objective-C. En direkt konsekvens av att olika språk används är att minneshanteringen för de båda plattformarna skiljer sig från varandra. Vid utveckling av Android-applikationer behöver inte programmeraren bry sig om allokering och frigörande av minne, det sköter Javas inbyggda skräpsamlaren (eng. *garbage collector*) i Dalvik om [AND2011]. I iOS måste minne allokeras och frigöras manuellt [BIL2009].

4.2 Start av applikation

iOS-applikationer startas alltid via en `main()`-funktion till skillnad från Android-

applikationer som startas med den `Activity` som bestämts som huvudaktivitet i *AndroidManifest.xml*. En fördel med Androids tillvägagångssätt är att applikationer kan göras modulära. En iOS-applikation består inte av flera olika komponenter på samma sätt som Android-applikationer kan göra, där de olika komponenterna kan startas helt oberoende av varandra.

4.3 Applikationers livscykel

En iOS-applikations livscykel var tidigare mindre komplex än en Android-applikations livscykel. Det berodde på att iOS inte hade stöd för *multitasking,* men från och med version 4.0 av iOS finns stöd för det. Applikationernas livscykler i de båda plattformarna skiljer sig ändå fortsättningsvis från varandra.

I en Android-applikation kan varje `Activity` ha sin egen livscykel medan iOS-applikationer endast har en livscykel per applikation. Om man jämför de olika tillstånden för en Activity i en Android-applikation och en hel applikation i iOS är de ganska lika, den enda skillnaden är att iOS har två tillstånd för applikationer som inte visas, *bakgrund* och *avbruten*, där *bakgrund* är det tillstånd som tillåter exekvering av kod. I Android tillåts inte vanliga aktiviteter exekvera kod i bakgrunden, en `Service` ska användas för det istället.

4.4 Distributionsarkiv

Både Android- och iOS-applikationer distribueras i en form av arkiv som innehåller alla filer som applikationen behöver. Båda två innehåller olika typer av resursfiler som till exempel bilder, ljudfiler och övriga rådata. Båda två måste också innehålla en definitionsfil i XML-format, *AndroidManifest.xml* för Android respektive *Info.plist* för iOS, annars startas inte applikationen. Utöver det innehåller iOS-arkivet applikationens exekverbara fil medan Android-arkivet innehåller en exekverbar fil med Java-klasser.

4.5 Designmönster

iOS och dess ramverk och bibliotek är starkt påverkade av de designmönster som togs upp i kapitel 3.2. Att försöka utveckla applikationer för iOS utan att använda dem är i princip omöjligt eftersom de finns med så starkt i de klasser som

används. Android har bara ett lika starkt designmönster: objektorienterad programmering. Förutom det finns det rekommendationer i utvecklingsmanualen för hur olika saker ska göras för att fungera bäst, som till exempel hur applikationer ska utvecklas för bästa möjliga prestanda [AND2011].

4.6 Användargränssnitt

I uppbyggnaden av användargränssnitt är skillnaderna ganska små. Applikationsfönstret byggs upp av rektangulära byggstenar i form av Androids `View` eller iOS:s `UIView`. Inuti de rektangulära rutorna kan knappar, textrutor, tabeller och liknande element placeras ut.

4.7 Kodexempel

Androids version av ”Hello, World!”-applikationen kan snabbt konstateras vara flera gånger kortare och åtminstone för en nybörjare lättare att förstå än motsvarande iOS-applikation. För det första innehåller Android-versionen endast en fil med källkod medan iOS-versionen innehåller fem filer. Att iOS-versionen innehåller så många filer beror till en del på ett av designmönstren i iOS, *MVC*, som säger att klasser ska delas upp i data-, gränssnitts- och kontrollklasser. Android-versionen i sin tur har all kod i den funktion som anropas då huvudaktiviteten startas.

5 Avslutning

I den här avhandlingen har jag jämfört applikationsstrukturen i de mobila operativsystemen Android och iOS. De faktorer som jag har tagit upp är start av applikationer, bakomliggande utvecklingsramverk, minneshantering, applikationers livscykel, distributionsarkiv, designmönster och användargränssnitt. Dessutom illustrerades jämförelsen med ett exempelprogram för vardera plattformen för att enklare kunna se skillnaderna.

Strukturen i Android skiljer sig klart från strukturen i iOS på många punkter. Det innebär att utvecklingen för de båda plattformarna också skiljer sig från varandra. Till exempel skiljer sig starten av applikationer i Android ganska mycket från iOS, applikationer i Android kan innehålla olika komponenter som kan startas

oberoende av varandra. En annan aspekt som skiljer sig mycket mellan plattformarna är minneshanteringen, man måste vara väldigt noggrann i iOS med allokering och frigörande av minne medan Android sköter om minneshantering helt automatiskt. Även utnyttjandet av designmönster skiljer sig markant mellan plattformarna. I iOS tvingas man i princip använda designmönster hela tiden, medan man i Android inte följer någonting annat än den objektorienterade principen.

För att avgöra vilken plattform som är bättre för en specifik applikation måste ändå en mycket mera detaljerad undersökning göras. Exempel på faktorer som kan påverka val av utvecklingsplattform är utvecklarnas tidigare erfarenheter, målgrupp för applikationen och om kod från andra applikationer ska återanvändas.

Litteraturförteckning

[AND2009]: Andrew S. Tanenbaum, Modern Operating Systems, Third Edition, 2009, Pearson Education, Upper Saddle River, NJ, USA

[BIL2009]: Bill Dudney, Chris Adamson, iPhone SDK Development, 2009, The Pragmatic Bookshelf, USA

[CHR2011]: Christopher Breen, The iPhone Pocket Guide, Fifth Edition, 2011, Peachpit Press, Berkeley, CA, USA

[DAV2009]: Dave Mark, Jeff LaMarche, Beginning iPhone Development: Exploring the iPhone SDK, 2009, Apress, USA

[ERI2008]: Erica Sadun, The iPhone Developer's Cookbook, Building Applications with the iPhone SDK, 2008, Pearson Education, Boston, MA, USA

[HAR2009]: Harry McCracken, Smart Phone OS Smackdown, 2009, PC World, Vol. 27, Issue 2, s. 54-58, USA

[JAM2011]: James Steele, Nelson To, The Android Developer's Cookbook, Building Applications with the Android SDK, 2011, Pearson Education, Boston, MA, USA

[MAR2011]: Margaret Butler, Android: Changing the Mobile Landscape, 2011, IEEE Pervasive Computing, January–March 2011, s. 4-7, Los Alamitos, CA, USA

[PEI2006]: Pei Zheng, Lionel Ni, Smart Phone and Next-Generation Mobile Computing, 2006, Morgan Kaufmann Publishers, San Francisco, CA, USA

[THE2007]: Theo Dunnewijk, Staffan Hultén, A brief history of mobile communication in Europe, 2007, Telematics and Informatics, Vol. 24, Issue 3, s. 164-179, USA

[AND2011]: Android Developers, http://developer.android.com/index.html, hämtad 26.2.2011

[ANN2008]: Announcing the Android 1.0 SDK, release 1, http://android-developers.blogspot.com/2008/09/announcing-android-10-sdk-release-1.html, hämtad 15.3.2011

[GOG2011]: Google’s Android becomes the world’s leading smart phone platform (Canalys research release: r2011013), http://www.canalys.com/pr/2011/r2011013.html, hämtad 26.2.2011

[HTC2008]: HTC - Press - 2008 - T-Mobile Unveils the T-Mobile G1 - the First

Phone Powered by Android, http://www.htc.com/www/press.aspx?id=66338&lang=1033, hämtad 26.2.2011

[IOSD2011]: iOS Developer Library, http://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html, hämtad 31.3.2011

[OHA2011]: Open Handset Alliance, FAQ, http://www.openhandsetalliance.com/oha_faq.html, hämtad 26.2.2011

Figur- och tabellförteckning