Agil användbarhetsutveckling för handhållna enheter
Androidlaboration

Per Lind

31 januari 2016

Innehåll

Inledning

Uppgiften för denna laborationen är att skapa en mobilapp som visar en listvy och en detaljvy i en adaptiv layout och som låter användaren ändra på innehållet i listan. På mobiler kommer vyerna visas en åt gången medans de på tablets kommer visas sida vid sida. Det ska finnas en actionbar med menyalternativ som låter användaren skapa nya poster, ta bort poster, ändra sortering, ändra ordning och visa hjälpdialog. Poster ska sparas persitent i en lokal SQLite databas, det ska vara möjligt att hämta ut sorterade listor ur databasen.

Syfte

Denna laborationen kommer vara förberedande för projektarbetet och kommer därför ta upp standard widgets från Android API:et [1]. Ni kommer få bekanta er med Android Studio och även sätta upp versionhantering där i.

Material

Vi kommer i denna kursen helt använda oss av den nya utvecklingsmiljön Android Studio, vilket är baserat på IntelliJ IDEA. Android är en mobilplatform skriven i Java så ni kommer behöva ha JDK 7 eller högre installerat. För att kunna göra laborationerna krävs att man har Android SDK. I SDK Manager så behövs som minimum dessa paket :

- SDK tools
- SDK platform-tools
- SDK build-tools (senaste rev.)
- SDK Platform (API 23)
- Support Library
- Support Repository
- Google USB driver

För att sedan kunna skapa en enhet att köra på emulatorn så kommer ni även behöva SDK platformen och en system image för den API versionen. För de som kommer använda emulatorn på Windows så rekommenderas också att ni hämtar och installerar HAXM för att snabba upp emulatorn. Ni borde skapa två virtuella enheter, förslagsvis en Nexus 5 och en Nexus 7 (2012 års version). För att byta skärmorientering på emulatorn använder man 7:an på numpad alternativt ctrl + F12. Det är rekommenderat att inte stänga av emulatorn mellan körningar av appen eftersom den tar lång tid att starta upp.

Android Studio

Android Studio är en ny utvecklingsmiljö som förenklar hanteringen av androidprojekt. Den nya android-projektvyn är anpassad för att ge en bra överblick av projektet och grupperar filerna på tydligare sätt än den klassiska projektvyn [2]. Android Studio har en bra layout editor som kan visa previews. Det är möjligt att visa previews för flera enheter samtidigt vilket ger en bra överblick av hur layouten anpassar sig till olika skärmstorlekar. Nedan listas några nyttiga Android Studio kommandon (för windows):

- **Ctrl + Mellanslag** auto-complete/förslag
- **Alt + Insert** Generera metod
- **Ctrl + Q** Visa javadoc
- **Ctrl + P** Visa parameterar för metod
- **Ctrl + D** Duplicera rad
- **Ctrl + Y** Ta bort rad
- **Alt + Enter** Project quick fix, rättar till kodningsfel
- **Ctrl + Alt + O** Optimize Imports, tar bort oanvända imports bland annat

Debugging

I Android Studio är det möjligt att debugga en applikation som körs i emulatorn eller på en android-enhet [3]. Det går bl.a. att sätta brytpunkter i koden för debug läge genom att klicka på den grå vänstra kanten på raden man vill sätta brytpunkt på. För att skriva debug meddelanden till systemloggen använder man *Log.d()*. Systemloggen kan ses i LogCat under fliken Android DDMS (Dalvik Debug Monitor Server), där går det även att filtrera ut loggar. Från DDMS är det även möjligt att ta screen captures och screen recordings.

Byggsystem

Androids nya byggsystem använder gradle [4]. Filen *build.gradle* i mappen *app/* innehåller viktiga värden så som *targetSdkVersion* och *minSdkVersion*. Det är även här man hanterar importering av java-bibliotek bland annat. För att lättare hantera sina gradle-inställningar kan man i Androd Studio öppna ett GUI för detta, *File -> Project Structure*.

Versionshantering

Det finns bra stöd för versionshantering i Android Studio och hittas under menyvalet *VCS*. För att skapa en git repository så väljer man *Import into Version Control -> Create Git Repository*. När versionshantering används i ett projekt kan man ifrån fliken *Changes* utföra diverse git-kommandon. När man skapat en ny git repository inuti Android Studio behöver filerna inkluderas i versionshanteringen genom att man högerklickar på *unversioned files* och väljer att lägga till dem. Sedan räcker det att göra en commit för att få in dem i repository:n.

Testning

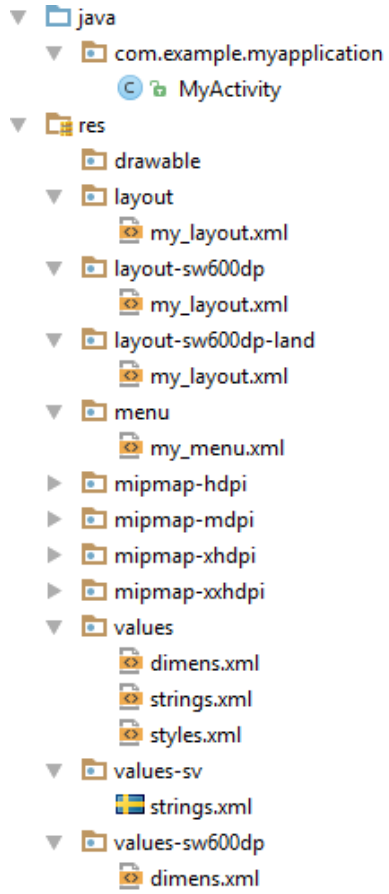
Android har stöd för två olika sorters tester, enhetstester (JUnit) och integrationstester (Android Instrumentation Tests). I Android Studio 1.1 var detta fortfarande en experimental feature, men under det senaste året har testningsramverken mognat och finns nu som support library [5].

Android Introduktion

Android är en mobilplattform som främst används av mobiltelefoner och tablets. En Android-app skrivs i Java och kompileras till en APK-fil (*Android Package*) för att kunna installeras av enheter som kör Android. Androidplattformen använder "principle of least privilige", dvs. att en app endast har tillgång till de komponenter som den behöver och inget mer. En app måste ha en manifest-fil, denna används av systemet för att berätta vilka komponenter som projektet består av. Appen kan i sitt manifest begära extra systemrättigheter (*permissions*) som t.ex att ringa samtal, läsa från extern lagringsenhet, använda kamera eller bluetooth. Av säkerhetsskäl måste dessa rättigheter godkännas av användaren.

Resources

Androids projektstruktur delar upp kod och resurser i mapparna java/ och res/. Oftast vill man använda olika resurser beroende på bl.a. skärmstorlek, skärmdensitet eller språk, detta görs genom att skapa resursmappar med postfix så kallade *qualifiers*. Det har tidigare funnits flera olika qualifiers för skärmstorlek, den nuvarande metoden är att sätta en *smallest width*. Till exempel så används sw600dp för att specifiera resurser till 7" och större skärmar, figur 1 . Det finns väldigt många olika qualifiers, se [6].



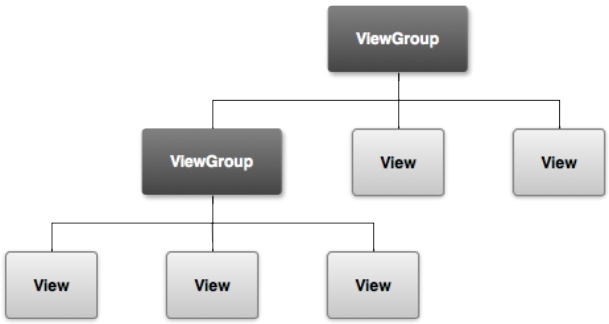
Figur 1: Androids projektstruktur

När man bygger projektet så genereras klassen *R.java* vilket används för att referera till resurserna[7]. Utöver den egna *R*-klassen går det även att referera till resurser ifrån Android API:et med klassen *android.R* [8].

Layout

Alla UI element ärver ifrån klassen View, ett användargränssnitt byggs upp av Views och ViewGroups, figur 2 . LinearLayout är en ViewGroup som listar Views på rad i en riktning. När man använder LinearLayout kan man sätta vikter för att avgöra hur mycket plats varje enskild View ska ta upp. För att undvika att behöva nästla LinearLayouts så finns RelativeLayout vilket låter en ge Views position relativt andra Views.

När man skapar en View så måste man sätta en höjd och en bredd, man kan antingen använda värderna match_parent eller wrap_content från ViewGroup.LayoutParams alternativt sätta ett exakt värde i dp. Position i layouter mäter man i dp (density-independent pixel) istället för pixlar och textstorlekar mäter man i sp (scale-independent pixel). Det är rekommenderat att låta UI element vara åtminstone 48dp höga och breda vilket på de flesta skärmar kan översättas till circa 9mm [9].



Figur 2: Illustration av viewgroup-hierarkin tagen från android dokumentationen

Komponenter

Appar består av komponenter, det finns fyra olika typer av komponenter i Android: *Activities*, *Services*, *Content Providers* och *Broadcast Receivers*. För att kommunicera mellan komponenter så använder man klassen *Intent*, dess tre vanligaste sysslor är att starta en activity, starta en service och skicka broadcast-meddelande till andra appar. Intent kan skicka med data med hjälp av metoden *putExtra*.

Activity

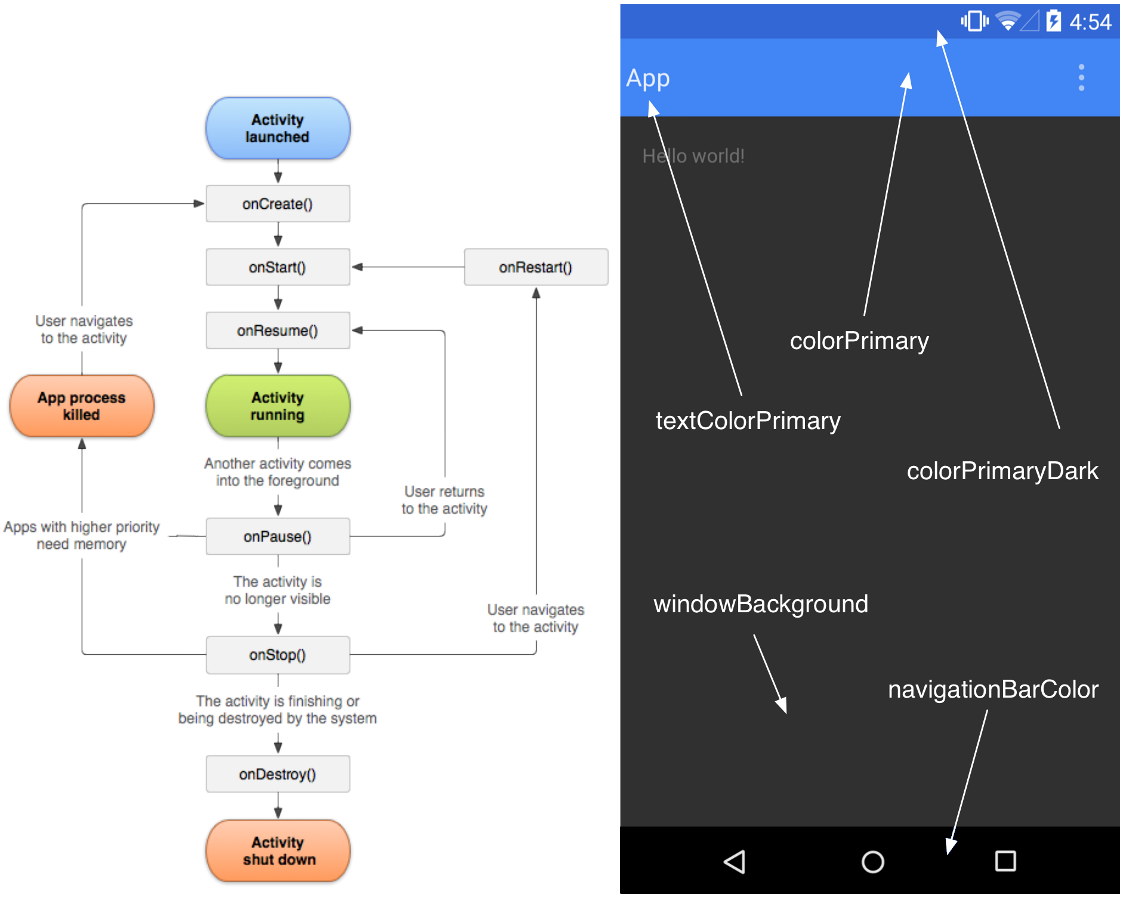
Vi kommer i denna laborationen fokusera på komponenter av typen *Activity* vilket representerar en vy i appen. Ingångspunkten i en app är vanligtvis den activity som är satt som launcher activity i manifestet och startas när man klickar på appikonen. För att starta en annan activity ifrån sin launcher activity så skickar man in ett Intent-objekt till metoden *startActivity*. För att kunna starta en activity måste den finnas med i dess apps manifest.

Activity implementerar specifika callback-metoder för olika event som är kopplade till dess livscykel, se figur 3. När man skapar en ny activity så initierar man den i *onCreate*. Metoden *setContentView* används för att koppla activityn till en layout-resurs. Man använder metoden *findViewById* för att hämta ut de widgetar från layouten som man vill ändra på dynamiskt i koden.

När man roterar skärmen så kommer activityn förstöras och sedan skapas om eftersom konfiguration har ändrats och den kan behöva läsa in nya resurser. Detta gör att views i layouten kommer förlora eventuellt tillstånd. För att kunna spara views tillstånd och återskapa dem så behöver man spara sina views dynamiska state i metoden *onSaveInstanceState()*. De sparade tillstånden kan man sedan återskapa i *onCreate* alternativt i metoden *onRestoreInstanceState*.

Fragment

Fragments representerar en del av en vy som kan placeras inuti en annan vy. En fragment kan bara visas inuti en Activities layout. Den har sin egna livscykel som till större del påminner om activities livscykel och är direkt relaterad till den[10]. Interaktion med fragment görs genom FragmentManager.
ListFragment är en subklass vars layout måste innehålla en ListView med id *@android:id/list*. ListView är en ViewGroup som visar en scrollbar lista utav Views. Den använder en AdapterView som mappar värden från ett dataset till Views i en layout.



Figur 3: Till vänster Activity lifecycle, till höger Material Theme

Lagring av data

Android har flera alternativ för att lagra data från en app: SharedPreferences, Internal Storage, External Storage och SQLite. I denna laboration kommer vi ta upp SharedPreferences och SQLite.

Tema

I Android 5.0 introducerades ett nytt tema, Material Theme. Detta temat är baserat på en omfattande designspecifikation [11]. Vi kommer i laborationen använda oss utav support biblioteket appcompat, dess tema Theme.Appcompat ärver ifrån det nya temat Material Theme. Temat inför bl.a. nya tema attribut för färger, figur 3 . Det är nu också möjligt att sätta tema för individuella views.

Support library

Som standard rekommenderas att man använder support library appcompat-v7, vilket innefattar support för Action Bar och Material Theme. Fördelarna med supportbiblioteken är att det går att använda nyare API:er på äldre versioner av android och att supportbiblioteken kan ha buggfixar. Man ska dock inte använda API:er från support library om man inte behöver. Att inkludera supportbiblioteken gör också så klart APK-filen (Android Package) större.

Bra länkar

Om API:er

Förklaring av de färdiga kodningsmallarna för activities
`http://developer.android.com/tools/projects/templates.html`

Google API:er så som Maps, Drive och Google+
`https://developer.android.com/google/index.html`

Två nya widgets som introducerades med API 21, RecycleView och CardView.
`https://developer.android.com/training/material/lists-cards.html`

Steg-för-steg ersätta ActionBar med Toolbar
`http://developer.android.com/training/appbar/setting-up.html`

Om design

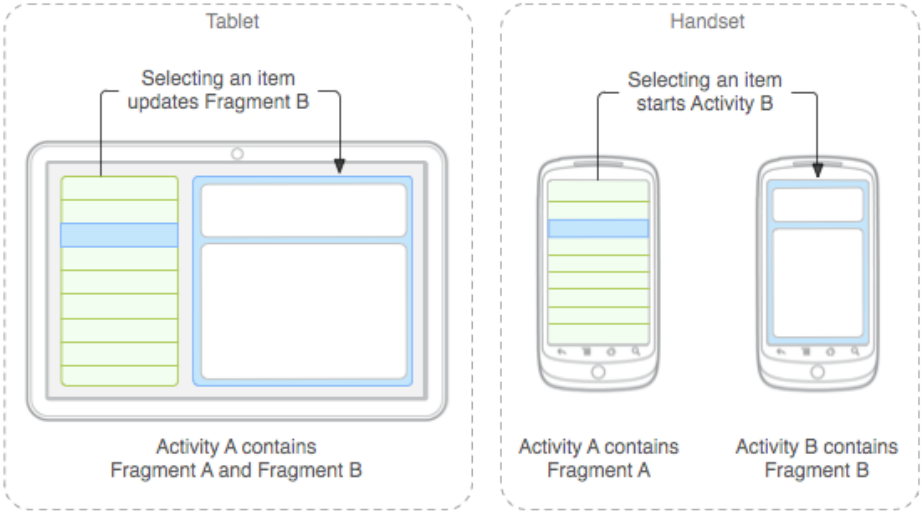
Bra blogpost om att använda material theme med support library
`http://android-developers.blogspot.se/2014/10/appcompat-v21-material-design-for-pre.html`

Presentationerna *Material Science* och *Material Witness* ifrån Google I/O 2014
`https://www.google.com/events/io/io14videos/`

Dokumentaion om Material Design
`https://developer.android.com/design/material/index.html`

Adaptiv layout

En adaptiv layout tar hänsyn till skärmstorleken och anpassar layouten utifrån den. Vi ska skapa en app som använder en adaptiv layout för att visa upp en lista i en huvudvy och detaljerna från posterna i en detaljvy. Skapa ett nytt projekt, låt *MinSdkVersion* vara API 19 och sätt *targetSdk* till API 23. Lägg till en *Master/Detail Flow* activity till projektet, detta är en kodningsmall som implementerar en adaptiv layout för att visa en lista. Koden för huvudvyn och detaljvyn ligger i var sin fragment, som vi kommer kalla *fragment a* och *fragment b* se figur 4 .



Figur 4: Exempel på en adaptiv layout med fragments från androids dokumentation

På mobiler kommer appen visa två separata activities, en för *fragment a* och en för *fragment b*. På tablets används en annan layout för *activity a* där *fragment a* och *b* visas sida vid sida.

Uppgift: Skapa en Master/Detail flow activity. Kör appen på en platta och kontrollera att *fragment a* och *b* visas i samma vy. Välj en post för att se att *fragment b* uppdateras. Pröva att rotera skärmen för att se att layouten visas korrekt i båda lägen. Testa sedan på en mobil och se att *fragment a* och *b* visas i var sin activity istället.

Uppgift: Lägg till versionshantering, skapa ett git repository för detta projektet och gör en commit. Kontrollera efteråt att er commit finns i *local history*.

Appbar och menyer

Efter att ha satt upp den adaptiva layouten så är det dags att lägga till lite knappar. I Android 3.0 introducerades widgeten action bar, vilken visar menyer tillsammans med uppåt-navigering samt appens titel och ikon. En activity eller fragment kan lägga till en *option menu* i action baren vilket som standard visas i en action overflow-lista. För att göra menyvalen alltid synliga i action baren som *action buttons* istället kan man sätta attributet *showAsAction* på menyalternativeten. Det är rekommenderat att endast låta de alternativ som används väldigt ofta vara action buttons.

Appbar och menyer fortsätter på nästa sida...

Toolbar

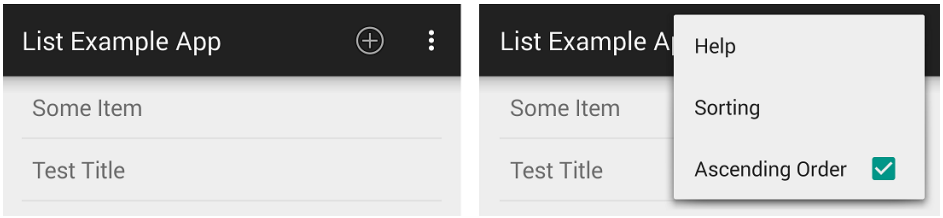
Det senaste året har fört med sig några ändringar, en av de större av dem måste ändå vara att action bar har i stort sett ersatts till fördel av den nya *Toolbar* widgeten. Anledningen är att action bar har utvecklats med åren och har därför olika implementation på olika plattformar, den har också stora begränsningar så som att material design inte kan användas på plattformar lägre än 5.0 Marshmallow. Support Toolbar däremot har samma implementation på samtliga versioner av Android och är mer flexibel. Det går bland annat att placera ut dem var som helst på skärmen och man kan placera ut flera stycken.

Ifall ni använder en äldre version av Android Studio så kommer er code template inte att generera Activities med toolbars. Enklaste sättet att rätta till detta är att följa denna guide [12].

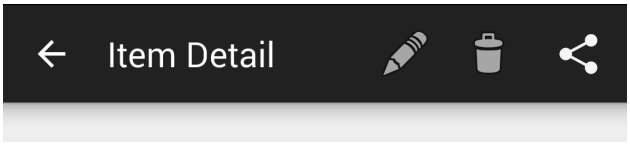
Uppgift: Verifiera att samtliga vyer har en toolbar, lägg till om det saknas.

Option menu

För att skapa och lägga till en meny i en activity behöver man skapa en *menu* folder i *res/*, högerklicka på *res* och välj *New -> Android resource directory* och välj *menu* som *Resource type*. Skapa därefter en ny *Menu resource file* för att skapa en options menu och lägg till menyvalen. Det är möjligt att ge menyalternativen en ikon, androids standardikoner går att hitta i *@android:drawable/*. Slutligen behöver man lägga till menyn i *onCreateOptionsMenu* med metoden *inflate* som läser in menyn. För att hantera menyval överskuggas metoden *onOptionsItemSelected*. Om man läser in menyn i en fragment så behöver man också sätta *hasOptionsMenu* till sant i *onCreate*.



Figur 5: Option menu för *fragment a*



Figur 6: Option menu för *fragment b*

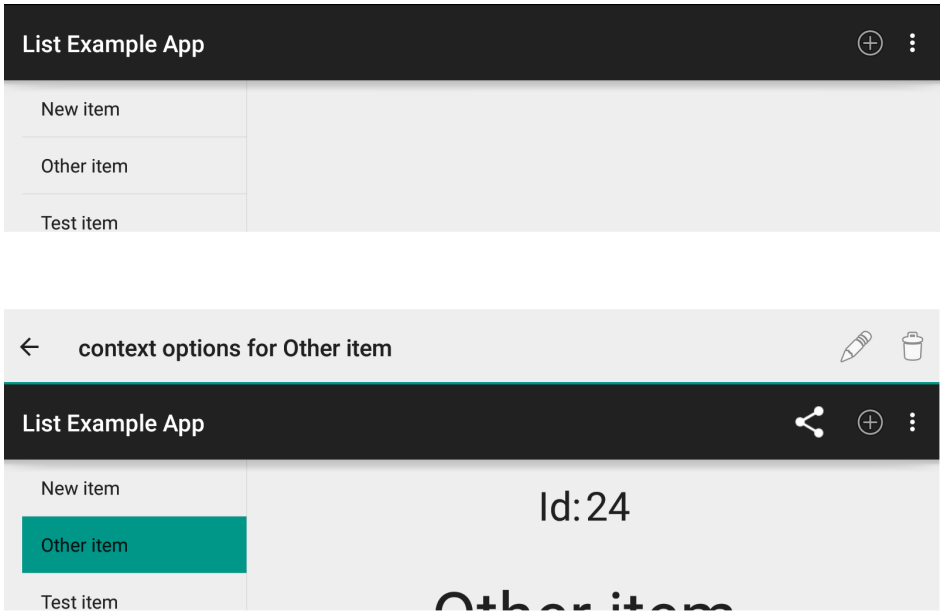
Uppgift: Skapa en meny för *fragment a* med alternativen: *add*, *sorting*, *ascending order* och *help*. Låt *add* vara en action button och ha övriga alternativen i action overflow, se figur 5.

Uppgift: Skapa en meny för *activity b* med alternativen: *delete* och *edit*, se figur 6.

För dessa uppgifterna räcker det att menyalternativen är klickbara.

Context menu

För att hantera kontextberoende menyval som att ta bort vald post i en ListView så kan man använda en *contextual menu*. Det finns två sätt att skapa en contextual menu i android, som en popup eller som en extra action bar. För att skapa en *contextual action bar* implementerar man *ActionMode.Callback* och anropar *startActionMode* [13].



Figur 7: Action baren för *activity a* på nedre bilden är en post vald.

Uppgift: Skapa en *contextual action bar* med alternativen *delete* och *edit*, se figur 7. Denna ska endast visas på plattor, när man använder mobil så visas dessa alternativen i en *option menu* istället, figur 6.

Lagring av data

Android har stöd för SQLite databaser vilket är ett av de smidigaste sätten att persitent lagra strukturerad data lokalt. Databaser skapade med *SQLiteHelper* är privata för appen som skapade dem. Koden för att skapa en databas och lagra datan av den egna typen Item finns redan i filerna *Datasource.java* och *Item.java*. Det är dags att byta ut DummyContent mot Datasource. En koppling till databasen borde öppnas när *fragment a* skapas och den bör stängas när fragmentet förstörs.

Uppgift: Skapa en instansvariabel av typen Datasource och ändra datatyp för ArrayAdapter:n till Item. Populera ListView:n med posterna från databasen istället för att använda *DummyContent*.

Lägga till post

I *fragment a* används en *ArrayAdapter* för ListView:n, för att lägga till en post kan man använda metoden *add* i ArrayAdapter. När ni gör detta bör ni även spara datan i databasen, spara posten med metoden *insertItem* i Datasource-klassen. Eftersom vi använder en ArrayAdapter men vi låter Datasource ta hand om sorteringen av listan så behöver vi hämta en ny sorterad lista ifrån databasen. Det är möjligt att undvika

detta och istället implementera en egen *sort*-metod för er adapter, men eftersom SQL kan göra sorteringen åt oss så går det lika bra att låta den göra det.
När innehållet i listan nu har ändrats behöver man uppdatera innehållet i ListView och anropar då metoden *notifyDataSetChanged*. Om man har en markerad position i listan och gör ändringar på listan behöver man uppdatera positionen för den valda posten så att inte fel post blir markerad.

Uppgift: När man klickar på *add* i action baren så ska en ny post läggas till i databasen och listan ska uppdateras.

Extra uppgift: Öppna en dialog och låt användaren fylla i värderna för den nya posten.

Extra uppgift: Implementera samma beteende för menyalternativet *edit*. Öppna en dialog och låt användaren ändra värderna för den valda posten.

Activities och fragments

Skicka data

För att *fragment b* ska kunna visa detaljerna för en post behöver *fragment a* skicka datan. När man använder en platta så är det *activity a* som skapar *fragment b* i callback-metoden *onItemSelected*. Den enklaste lösningen är att låta *onItemSelected* ta den relevanta datan som argument och sedan skicka datan till *fragment b* med *setArgument*.
Använder man en mobil så är det istället *activity b* som skapar fragment b. I detta fall så behöver *activity b* först få datan från *activity a* innan den kan skickas till fragmentet med *setArgument*. För att skicka data mellan activities använder man *putExtra*-metoder för att lägga till data till intent-objektet.

Uppgift: Skicka datan om en post från *fragment a* till *fragment b*. Testa att data skickas till fragmentet när man använder mobil och när man använder platta.

När man har fått datan till fragment b så ska det visas upp. Skapa en ny layout resource i res/layout. Lägg till ett par TextViews och alternativt även en RatingBar. Använd den nya layouten i fragment b, man sätter layouten med setContentView. Metoden findViewById gör att man kan hämta ut widgetarna i koden och sätta deras värden.

Uppgift: Skapa en egen layout för *fragment b* och använd den för att visa upp en post.

Lämna svar

För att kunna ta bort en post ifrån databasen så använder man postens id. När man använder layouten för mobiler och vill ta bort en post så behöver man skicka tillbaka *id* för den posten från activity b till activity a. För att få tillbaka data använder man *startActivityForResult* istället för *startActivity* och överskuggar metoden *onActivityResult*. Den startade activityn kallar på metoden *setResult* för att sätta resultatet som kommer returneras när activityn stängs.

Uppgift: När man klickar på *delete* i menyn i *activity b* så ska postens *id* skickas tillbaka till *activity a*.

Ta bort post

För att kunna ta bort en post ur databasen behöver man dess id och för att ta bort den ur arrayadaptern så behövs dess position. I datasource använder man metoden delete och i arrayadapter metoden remove. Datatypen Item har metoden getId för att hämta ut id-värdet. Beteendet för att ta bort en post kommer skilja sig för mobiler och plattor.

I fragment a finns metoden *setActivatedPosition* som man kan använda för att lagra positionen för den valda posten när man använder en platta. När man har tagit bort posten ska ingen post längre vara markerad och fragmentet som visar den borttagna posten ska självfallet också tas bort. Detta kan man enkelt hantera med FragmentManager genom metoderna addToBackStack och popBackStack.

För mobiler så måste man däremot hantera menyalternativet i activity b istället och skicka tillbaka id-värdet till activity a, vilket togs upp i den förra uppgiften. Skapa en public metod i fragment a som tar id som argument och tar bort posten ur databasen och listan. För att anropa metoden ifrån activity a hämtar man ut fragment a ifrån FragmentManager med findFragmentById.

Uppgift: På platta när man väljer *Delete* så ska posten tas bort från databasen, listan ska uppdateras, ingen post ska vara markerad och fragment b ska tas bort.
Uppgift: På mobil när man väljer *Delete* så ska activity b stängas, posten ska tas bort från databasen och listan ska uppdateras.

ListView

Sortering och ordning av listan

När man trycker på menyalternativet för sorting i ItemListFragment i *activity a* så ska en dialog visas. Ifrån dialogen ska det vara möjligt att välja vilken kolumn som sorteringen av listan ska använda. Använda en DialogFragment för att hantera dialogen, se [14]. Som i exemplet i länken så kan ni använda en AlertDialog.Builder för att skapa dialog med radio buttons för sorteringsalternativen.

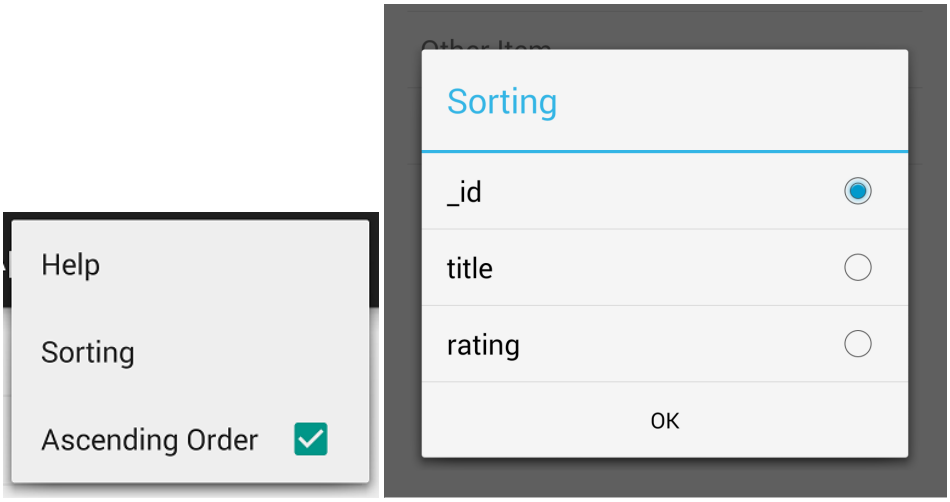
Det valda värdet borde lagras persitent med sharedPreferences, använd *getPreferences* för att hämta sharedPreferences i en fragment. SharedPreferences sparar privata key-value par för appen lokalt på enheten.

För inställningar som påverkar hela appen borde man skapa en PreferenceActivity istället för att hantera dom direkt i den öppna activity:n, i androids dokumentation finns en flow chart över när man borde använda en PreferenceActivity [15].

För menyalternativet *order ascending* så räcker det att använda en checkbox i overflow menyn, för att göra det så behöver man bara sätta attributet checkable till true. Detta värdet borde också sparas som en sharedPreference precis som sorteringen gjorde.

Uppgift: Menyalternativet *sorting* ska låta användaren välja vilken kolumn datasetet ska sorteras på. Vald kolumn ska lagras persitent, själva sorteringen hanteras av klassen Datasource.

Uppgift: Menyalternativet *ascending order* ska visas som en checkbox i action barens overflow. Checkboxens värde ska lagras persitent, själva sorteringen hanteras av klassen Datasource.



Figur 8: Till vänster menyval för sortering och ordning av lista och till höger dialog för sortering

Utseende och tema

Nu när all funktionallitet är på plats är det läge att snygga till appen och ge den ett eget tema. Appens tema finns i styles.xml och innehåller attirbut för att ändra layouterna. Dimens.xml används för att bl.a. spara appens egna marginaler, padding och textstorlekar.

Uppgift: Sätt egna färger på temat, appcompat-v7 använder Material.Theme som default vilket bl.a. har attributen colorPrimary, colorPrimaryDark och colorAccent, se figur 3 . Välj färger ifrån Material Design specen [16].

Uppgift: Sätt fontstorlek på samtliga TextViews med Material temats textstorlekar, se [17].

Extra Uppgift: När man kör Android *API 5.0* så används den satta colorAccent:n för att markera vald post i ListView:n. Skapa en StateListDrawable som ritar ut en bakgrund med samma färg när den är i activated state. Använd den nya drawable resursen som activatedBackgroundIndicator.

Extra uppgifter

Förbättra prestandan i ListView

Skapa en ViewHolder och hanterar listans View objekt. Lägg till ViewHoldern i AdapterView:n [18].

Extra uppgift: Optimera scrollningen av listan genom att lägga till View Holder-mönstret.

Hjälpdialog

För att skapa en dialog kan man använda en AlertDialog.Builder i en DialogFragment.

Extra uppgift: Menyalternativet *help* ska visa en dialog med en hjälptext och en knapp för att stänga dialogen.

Animering

Det är möjligt att lägga till animationer för fragment transitions. Skapa ObjectAnimators och spara dem i mappen res/anim/. Använd setCustomAnimation för att animera övergången när man lägger till och tar bort fragments.

Extra uppgift: Animera övergången när man lägger till eller tar bort fragments.

Floating labels

Floating labels fungerar som så att när man väljer ett textfält med en placeholder text istället för att ta bort texten så flyttas den till ovanför fältet och fungerar då som en label istället.

Extra uppgift: Se till att ni använder support design library. Gör om era textfält till att använda floating labels.

Floating Action Button

En av de mest omtalade ny komponenterna i Material Design måste vara Floating Action Button, en rund knapp som ligger ovanför övriga användargränssnittet och kastar en mjuk skugga bakom sig. Det är en komponent som är tänkt att användas för den viktigaste actions, det låter en flytta ut denna primära handlingen från er toolbar till en mer tillgänglig position i layouten. Gmail använder till exempel en floating action button med ett plustecken för sin primära action, att skriva nytt mail.

Extra uppgift: Se till att ni använder support design library. Gör om *lägg till post* action till en floating action button.

Share action

Man kan låta användaren dela en post med andra genom att använda en share action provider. Pågrund av en bugg som inte låter contextual action provider ha en ShareActionProvider widget som menu item så kan ni lägga menyalternativet *share* i en option menu istället. Skapa en share action, genom att använda en

ShareActionProvider [19]. Komihåg att uppdatera sharing intent när vald post ändras. Det finns ett exempel ni kan utgå ifrån, *File-> Import Sample -> ActionBarCompat-ShareActionProvider*.

Extra uppgift: Lägg till en *share action*. Skapa en ny *option menu* med endast en menu item med attributet *actionProviderClass* satt till "android.widget.ShareActionProvider". Ge menyalternativet en shareIntent med datan för den valda posten.

Filtrera listan

Det enklaste sättet att filtrera ut data när man använder SQLite databaser är att göra filtreringen i databasfrågan. Skapa en metod som gör detta i klassen Datasource. Lägg till ett menyval som låter användaren filtrera innehållet. Alternativt lägg till ett sökfält i action baren som användaren kan filtrera ut poster med.

Extra uppgift: Lägg till stöd för att göra filtrerade listor, välj en kolumn och ett värde att filtrera på.

Egen listview layout

Hittils har vi endast använt oss av den färdiga layouten android.R.layout.simple_list_item_activated_1. Låt oss ersätta den med en egen layout-fil och mappa värderna till olika element i layouten, använd en SimpleAdapter.

Extra uppgift: Skapa en egen layout för View-objekten i ListView.

Snackbar

Extra uppgift: Ge användaren feedback på deras actions. Lägg till ett ångra alternativ i snackbaren när man tagit bort en post.

Swipe action i listan

Extra uppgift: Tillåt användaren att ta bort en post genom att dra den åt sidan med en swipe.

Coordinator layout

Extra uppgift: Ändra utseende på toolbar när man scrollar layouten, använd en AppBarLayout i en CoordinatorLayout och kombinera gärna med en CollapsingToolbar.

Redovisning

Krav för godkänt

Visa upp appen på en mobil och på en platta. Appen måste uppfylla nedanstående krav för att bli godkänd.

- Ska visa lista och detaljvy på både mobil och tablet med *Master Detail Flow*-pattern
- Ska kunna lägga till, redigera och ta bort poster ifrån privat databas
- Ska kunna ställa in hur listan ska sorteras
- Eget *Material Design*-tema
- Ska kunna spara app-inställningar persistent
- Ska ha minst en dialog
- Ska ha minst en action button

Referenser

[1] API Guides, hemsida http://developer.android.com/guide/index.html

[2] Android Studio Overview, hemsida https://developer.android.com/tools/studio/index.html

[3] Debugging, hemsida http://developer.android.com/tools/debugging/index.html

[4] Build System Overview, hemsida https://developer.android.com/sdk/installing/studio-build.html

[5] Best Practices for Testing, hemsida http://developer.android.com/training/testing/index.html

[6] Providing Resources, hemsida http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources

[7] Accessing Resources, hemsida http://developer.android.com/guide/topics/resources/accessing-resources.html

[8] android.R, hemsida https://developer.android.com/reference/android/R.html

[9] Metrics and Grids, hemsida http://developer.android.com/design/style/metrics-grids.html

[10] Fragments, hemsida http://www.google.com/design/spec/material-design/introduction.html

[11] Material Design, hemsida http://www.google.com/design/spec/material-design/introduction.html

[12] Setting Up the App Bar, hemsida http://developer.android.com/training/appbar/setting-up.html

[13] Contextual Action Bar, hemsida http://developer.android.com/guide/topics/ui/menus.html#CAB

[14] DialogFragment, hemsida http://developer.android.com/guide/topics/ui/dialogs.html#DialogFragment

[15] Settings, hemsida http://developer.android.com/design/patterns/settings.html

[16] Color, hemsida http://www.google.com/design/spec/style/color.html#color-ui-color-application

[17] Typography, hemsida http://www.google.com/design/spec/style/typography.html#typography-standard-styles

[18] View Holder, hemsida http://developer.android.com/training/improving-layouts/smooth-scrolling.html#ViewHolder

[19] Share Action, hemsida http://developer.android.com/training/sharing/shareaction.html