Kommunikation och Användargränssnitt - Androidutveckling

Andreas Valter Fredrik Lindner Karl-Johan Krantz Jonas Strandstedt
Jonas Zeitler

July 17, 2014

1 Syfte

Denna laboration är en introduktion till Androidprogrammering för att ni ska ha möjlighet att göra en mobilapp som projekt. Ni kommer även att få en introduktion till hur man använder enklaste möjliga databas tillsammans med Android, det möjliggör applikationer med mer avancerade data.

2 Inledning

Uppgiften är att göra en "Att göra-lista" med möjlighet att lägga till och ta bort uppgifter.

3 Installation

För att installera Android Studio gå in på http://developer.android.com/sdk/installing/studio.html och välj "Download Android Studio Beta with the Android SDK". Utvecklingsmiljön är en färdigkonfigurerad version av IntelliJ IDEA och finns till Windows, Linux och Mac.

Packa upp zip-filen till D: och öppna Android Studio från från "bin/filnamn.exe". Första gången du öppnar Android Studio kommer den fråga var du vill placera dess workspaces. Vi föreslår att du väljer en mapp som finns på "privat_stuff", vi kommer att använda "H:/Workspace".

3.1 Skapa en Androidemulator

För att kunna köra en Androidapplikation behövs en Androidenhet. Eftersom vi inte utgår från att alla har en Androidtelefon/-platta föreslår vi att installera en Androidemulator. "Tools -> Android -> Android Virtual Device Manager / AVD Manager", välj sedan "Create...".
AVD Name: MyDevice
Device: Nexus 5
Target: Android 4.4 - API Level 19
Emulating Options: Use host GPU
Lämna resten som default.

3.2 Skapa Projektet

Nu har du startat Eclipse samt skapat en enhet. Nu är det dags för det roliga, koda!
Välj "File -> New -> Android Application Project" och följ instruktionerna. Fyll i inställningarna så att de ser ut som följer:
Application Name: MyToDoList
Kryssa i "Phone & Tablet" och välj
Minimum Required SDK: API 15: Android 4.0.3 (IceCreamSandwich)
Välj sedan att du vill ha en "Blank Activity", och ge den namnet "MainActivity".

Fortsätt genom att använda defaultinställningarna på resten av valen och avsluta med "Finish".
Kör din Androidapplikation genom att trycka på "Play" knappen i menyn eller Ctrl+F11, och kryssa sedan i "Launch emulator" och välj "MyDevice" som du skapade tidigare. Notera att Androidemulatorn tar några minuter att starta första gången. Stäng inte av emulatorn med krysset mellan körningar för då tar det lång tid att starta varje gång.

4 Todo-app

Vi kommer inte i det här kompendiet att gå in på djupet i hur Android fungerar. De som arbetat med iOS tidigare kommer att märka att det skiljer sig en hel del och att till exempel vyer hanteras annorlunda. Fokus på labben kommer vara att arbeta i Java och nödvändiga Androidspecifika komponenter kommer att beskrivas nedan.

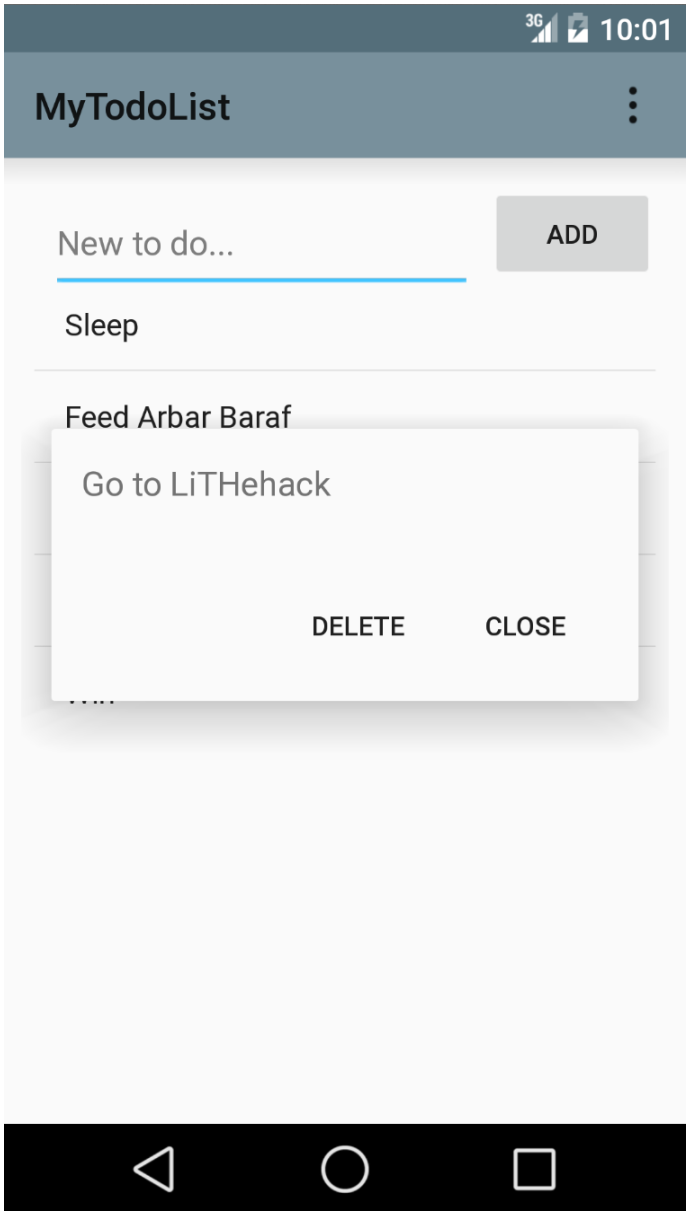

Figure 1: Exempel på hur en färdig ToDo-app kan se ut.

4.1 Debug prints

För den som är van att arbeta med textbaserade program är det vanligt att göra “debug prints” för att kontrollera att värden i variabler är korrekt samt se att funktioner blivit anropade etc.

Android Logcat, där loggarna du gör syns, ska starta samtidigt som du startar din emulerade telefon. En utskrift i LogCat görs genom att först ge ett meddelande en tagg som identifierar dess typ. Exempelvis "databas", "info" eller "debug". Vi kommer använda taggen "TodoLog". När LogCat är öppnad behöver vi skapa ett filter, för Android kommer att spotta ut väldigt mycket meddelanden och vi är just nu enbart intresserade av "TodoLog".

Klicka på rullgardinsmenyn till höger där det står något i stil med "app: com.example.lithehack.mytodolist", och välj sedan "Edit FIlter Configuration". Tryck sedan på plus-tecknet i fönstret som dyker upp. Ge ditt filter ett lämpligt namn, och i rutan "by Log Tag (regex)" skriver du sedan "TodoLog". Klicka på OK, och i samma rullgardinsmeny som tidigare, välj ditt nyskapade filter.

Testa att det fungerar genom att logga i funktionen onCreate.

```
Log.d("TodoLog", "Programmet startat");
```

4.2 Visa en lista

För att kunna skapa en lista med saker att göra är det lämpligt att först skapa den grafiska layouten för listan. Det grafiska gränssnittet ligger i filen "*res / layout / activity_main.xml*".
Filen bör ha öppnats när du skapade ditt nya projekt. Om den inte gjorde det, leta upp den i "Project"-vyn och dubbelklicka på den.

Antingen får du nu upp det grafiska gränssnittet för att redigera en vy, eller så får du upp det textbaserade gränssnittet. Oavsett vilket är det gott så, det finns en god poäng med att vara bekant med båda. För att växla mellan de två vyerna kan du byta flik strax under och till vänster om telefonen som syns. Alternativen som finns där är "Design" och "Text". I det som kallas "Design" kan du direkt redigera gränssnittet genom att dra in olika element ifrån "Palette" och i det som kallas "Text" redigera XML-koden.

Textfältet "Hello World!" kan du bara ta bort genom att antingen markera det direkt eller markera det i listan "Component Tree" till höger, och trycka på delete.

Skapa en lista genom att dra in elementet "*Container -> ListView*". För att våran aktivitet i appen ska kunna nå listan så måste den få ett lite annorlunda "id" än vad den har nu. I "*Properties*" finns det ett fält som heter "Id". Ändra innehållet i det till "*@android:id/list*". Detta är lite speciellt när det kommer till just en lista och vi kommer senare att behandla id på ett mer konventionellt sätt när det kommer till knappar och textfält.

Nu kan du återgå till koden i filen *MainActivity.java*, som innehåller källkoden för huvudaktiviteten i din app. Då vi har valt att skapa en lista i vår Activity, så gör vi bäst i att ärva från ListActivity istället för Activity i vår klass. Ändra så att MainActivity ärver ifrån ListActivity istället för Activity.

Android Studio vet inte automatiskt vad en ListActivity är och markerar detta som ett fel i koden. Importera klassen genom att klicka på den röda lampan som dyker upp då du ändrar detta och välj "Import class". Du kan också ställa markören i det rödmarkerade ListActivity och trycka Alt+Enter för att få upp denna meny.

För att göra en enkel lista att testa ditt grafiska gränsnitt med, så behövs det lite data att visa. Ge din klass en privat datamedlem som är en String ArrayList och döp den till *testData*. ArrayList är i syntaxen ganska lik Vector, som du nog är bekant med sedan tidigare. Deklarera även en privat datamedlem av klassen ArrayAdapter som tar typen String och döp den till *adapter*.

I metoden onCreate, som anropas direkt efter appen har startats, kan du instansiera din String ArrayList med lite testdata. Här gör du som du vill, ett sätt är bara loopa igenom arrayen och lägga in en siffra eller textsnutt för varje index.

Nu måste vi koppla denna data till vårt grafiska gränssnitt. Detta är inte helt intuitivt till en början, så ni får lite hjälp med det:

```
adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1 , testData);
setListAdapter(adapter);
```

Det som sker här är att vi skapar en s.k ArrayAdapter som kopplar ihop datan som är lagrad i *testData* med det grafiska gränsnittet. Vi anger vilken datatyp det är (String) och var datan är lagrad (testData). Tillslut tilldelar vi adaptern till aktiviteten via *setListAdapter*.

Om allt har gått rätt till så bör nu appen kunna startas genom att trycka på Run. Appen bör startas i Android emulatorn. En lista, som förvisso är rätt tråkig och statisk, bör nu visas!

Uppgift: Gör en lista i det grafiska gränssnittet. Skapa en ArrayList av typen String, initialisera den med lite data och koppla ihop den med en ArrayAdapter, också av typen String.

4.3 Interagera (click event) med en lista

Nu är det dags att få igång interaktionen med listan. Eftersom vår *MainActivity* ärver av *ListActivity*, så är det ganska enkelt att komma igång med.

För att skapa en event listener som lyssnar på interaktioner med listan ska vi överlagra metoden *onListItemClick* från *ListActivity*. Här kan vi använda Android Studio för att generera en metodsignatur. Ställ markören någonstans i klassen *MainActivity* (men inte i någon metod), börja skriva *"onListItemClick"* och tryck ctrl+space. Android Studio föreslår då att infoga en överlagrad variant av *onListItemClick* från *ListActivity* med alla funktionsparametrar som behövs.

OBS! Notera att du behöver ändra så att de två sista argumenten är konstanta, dvs *final int* respektive *final long*. Detta behövs senare för att ha tillgång till dem i en lyssnare.

Inne i metoden kan vi nu bestämma vad som ska hända när en användare klickar på en rad i listan. Enklast just nu är förstås att bara logga till LogCat.

När du nu kör din app igen, så bör LogCat visa din text när man trycker på en post i listan.

Uppgift: Implentera onListItemClick och skriv ut någon text genom LogCat för att se att interaktionen med listan funkar korrekt.

4.4 Visa en popup

Nu när vi vet att det funkar att klicka på en post i listan, så kanske det kan vara roligare att få upp ett meddelande i appen istället för LogCat.

Tidigare såg vi att man enkelt kunde utforma sitt grafiska gränssnitt med den grafiska editorn. Det kan man använda för mer eller mindre allt grafiskt arbete, men när det kommer till enklare saker som tex dialogrutor kan det vara lämpligt att skriva de direkt i koden.

För att göra detta använder vi oss av en *Builder* till komponenten *AlertDialog*. *AlertDialog* visar en popup-dialog anpassad för saker som att bekräfta en återgärd eller visa information om en post.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

Vi kan nu lägga till saker i våran builder. Anropa metoden setMessage med en sträng som argument för att sätta ett textmeddelande som dyker upp när man klickar på en post.

Vi har ännu inte skapat våran *AlertDialog*, men det gör vi nu genom att anropa

```
AlertDialog dialog = builder.create();
```

Vi kan visa dialogrutan genom *dialog.show();*

Kör appen och se hur du nu får en popup ruta med texten när du klickar på en post!

Uppgift: Använd dig av *AlertDialog* och dess tillhörande *Builder* för att skapa en dialogruta som visas när man klickar på en post i listan.

5 Ta bort poster

Låt oss nu försöka lägga till funktionalitet för att kunna ta bort en post ifrån listan. Metoderna *setPositiveButton* och *setNegativeButton* på builder-objektet ger tillgång till två knappar som man kan styra hur man vill. Båda två metoderna tar två argument; en sträng för texten på knappen och en *OnClickListener*. Här kan du antingen anropa en lyssnare som du skapat någon annanstans, eller helt enkelt skapa den direkt i metodanropet genom

```
new DialogInterface.OnClickListener() { }
```

Inuti denna lyssnare så måste metoden *onClick(DialogInterface, int)* finnas. Det är alltså här inne som funktionaliteten för att kunna ta bort en post ifrån listan skall implementeras.

För att ta bort en post ifrån listan så behöver du anropa metoden *remove* på *adapter*. remove tar i det här fallet ett objekt av typen String som argument, då ArrayAdapter:n har blivit deklarerad för att hantera just String. Här vill du nu plocka bort den aktuella posten i listan genom att hitta den i testData. I argumenten för *onListItemClick* så finns den aktuella postens index i konstanten *id*.
Om du gjort rätt så kommer posten tas bort från både testData och adapter, vilket kommer resultera i att listan i det grafiska gränssnittet uppdateras korrekt.

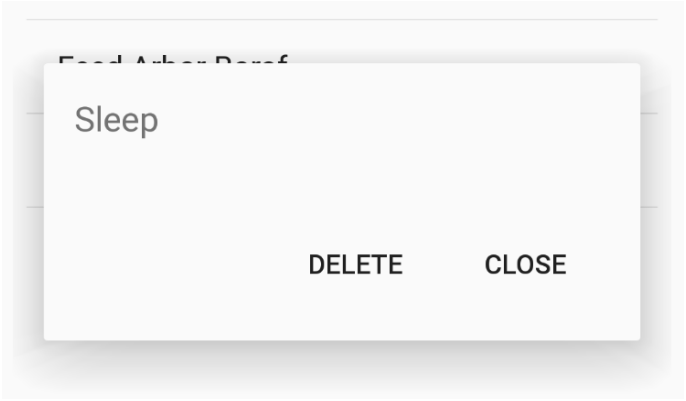

Figure 2: AlertDialog med möjlighet att ta bort den aktuella posten i listan.

Knappen för "Ok" behöver inte göra någonting, så hela onClick-metoden kan bara vara tom.

Uppgift: Skapa två knappar i dialogrutan, en som bara är "Ok" och inte gör något mer än att stänga rutan, och en "Delete" som tar bort den aktuella posten från listan.

5.1 Knappar och textfält

Det hade naturligtvis varit trevligt med möjligheten att lägga till nya poster i listan. För att göra det behöver vi en knapp för att lägga till, och ett textfält för att skriva in namnet.

Gå tillbaka till ditt grafiska gränssnitt (*activity_main.xml*).
Lägg nu till ett textfält ("Text Fields -> Plain Text") genom att dra och släppa. Lägg det så att två textrutor dyker upp där det står "alignParentLeft" och "alignParentTop". Lägg sedan till en knapp ("Widgets -> Button"). Placera knappen så att liknande rutor dyker upp, som säger "alignParentTop" och "alignParentRight". Justera nu storlekarna på både textfältet och knappen, så att textfältet får rutan "toLeftOf=button0" eller motsvarande, och knappen "above=@android:id/list". Till sist, justera storleken på listan så att rutan "below=editText0" eller liknande dyker upp.

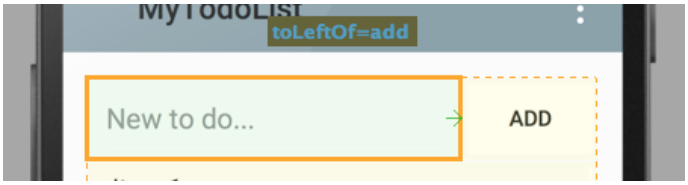

Figure 3: Rutor som dyker upp då "views" flyttas och ändrar storlek.

Nu har vi skapat en layout där objektens storlek och position är beroende av varandra. Detta är givetvis inget krav, men skapar tydliga linjer i gränssnittet, vilket får det att se mindre rörigt ut. Ge dem ett lämpligt *id* i *Properties*, tex "add" för knappen och "name" för textfältet. Detta är oftast enklast att ändra i "text"-läget. Kom ihåg att det måste vara på formen *@+id/ditt-namn-här*.

För att göra det lite lättare för användaren att hitta textfältet kan det vara bra att lägga till en så kallan *hint*. Gör det genom att markera textrutan i den grafiska editorn, och leta sedan upp

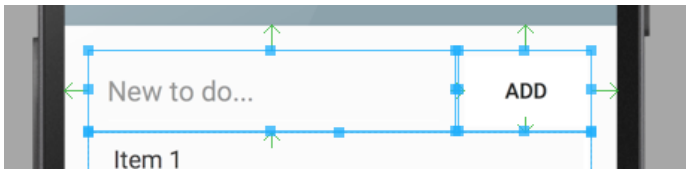

Figure 4: Elementen positions- och storleksberoenden.

fältet "hint" i "properties"-panelen. Skriv något lämpligt! Ändra texten i knappen på liknande sätt, genom att förändra fältet "text".

Om du testkör din app nu, så bör du se både knappen och textfältet som det ser ut när du designar ditt grafiska gränssnitt, men de gör inget än så låt oss fixa det.

För att kunna interagera med knappen så behöver vi lägga till lyssnare. Detta gör vi i onCreate-metoden. Skapa först knappen genom:

```
Button button = (Button)findViewById(R.id.add);
```

Nu kan vi anropa metoden *setOnClickListener* på *button*. Som tidigare kan vi skapa en lyssnare direkt i metodanropet

```
new View.OnClickListener() { }
```

Som tidigare behövs en överlagrad metod *onClick(View)* i lyssnaren (av typen *public void*). Innan vi kan definera vad knappen ska göra när användaren trycker på den, så måste vi identifiera att användaren har tryckt just på våran knapp. I *onClick* så har vi fått med en View som har den tillhörande metoden *getId()* för att hämta id på det elementet som har aktivierats. Se till så att det överensstämmer med id på våran knapp genom att jämföra med *R.id.add* (dvs, om knappen är döpt till "add").

Nu måste vi hämta ut informationen som är inskrivet i textfältet! På samma sätt som knappen skapades ovan så kan vi skapa ett textfält, som är av typen *EditText* istället för *Button*. Metoden *getText()* hämtar ut text från ett textfält, dock är det inte av typen String. Fixa det!

Sen är det bara att lägga in texten i form av en String i *adapter*. Det är väldigt likt det vi gjorde tidigare när vi tog bort elementet ur *adapter*...

Uppgift: Skapa en knapp och ett textfält. När man trycker på knappen skall texten som är i textfältet läggas till i listan.

6 Databaskoppling

Hittills har vi endast lagrat data i en ArrayList. Den tas naturligtvis bort så fort programmet stängs ned. Ett smidigare sätt att lagra data för en Todo-lista är att lagra allt i en liten SQL databas. All kod för att skapa en databas och lagra en egen klass i den är redan skriven, och finns i filerna *MySQLiteHelper.java*, *TodosDataSource.java* och *Todo.java*.

Här är ett bra ställe att klippa navelsträngen, nedan ges bara de metoderna som kan vara användbara för att få det att funka med databasen. Allt är redan färdigt i de filerna som skickas med så

inget behöver ändras där, men läs igenom dem för att förstå hur det hänger ihop. Försök komma så långt ni kan!

Datamedlemmar: ArrayList av typen *Todo*, ArrayAdapter av typen *Todo*, TodosDataSource
TodosDataSource metoder: open(), getAllTodos(), createTodo(), deleteTodo()

Uppgift: Skapa en Todo-app med koppling till en databas.

Lycka till!