NTNU
Kunnskap for en bedre verden

Datalagring

Mildrid Ljosland, Institutt for datateknologi og informatikk (IDI), NTNU
Lærestoffet er utviklet for faget IFUD1042 Applikasjonsutvikling for Android

Resymé: Å kunne hente fram data som brukeren (eller andre) har framskaffet, er nyttig i de fleste applikasjoner. Vi skal se på tre måter det kan gjøres på: Via "preferences", ved å lagre til fil samt ved bruk av database. Læreboka behandler dette i kapitlene 16, 17 og 18.

I Android har vi flere måter vi kan lagre data på. Måten vi gjør det på, vil avhenge av hvor mye og hvilken type data vi har. Hvis vi har enkle data av typen «nøkkel=verdi» og et begrenset antall av dem, vil ofte preference-mekanismen være det greieste å bruke.

Har vi derimot ustrukturerte data, gjerne ren tekst, vil det som oftest være best å bruke vanlige filer.

Har vi struktur, men for mye/komplisert til at preference er aktuelt, kan vi bruke en database. Eller vi kan lage en xml-fil og bruke en XML-parser, for eksempel en PullParser.

De tre første mekanismene blir behandlet i denne leksjonen, mens å lese xml-filer ikke behandles i dette kurset. Hvis du er interssert i denne metoden, se [PullParser].

7.1 Preference-mekanismen

Når man har en situasjon der man ønsker at bruker skal kunne stille inn ulike preferanser eller enkle verdier, for eksempel valg av språk, og ønsker at det skal huskes til neste gang applikasjonen startes, er Preferences fin å bruke. Android har både klassen PreferenceActivity og klassen PreferenceFragment, men PreferenceActivity er «deprecated» og anbefales ikke brukt.

Android sin Preference-mekanisme baserer seg på den abstrakte klassen Preferece, som har diverse subklasser. En av subklassene er PreferenceGroup, som er abstrakt og har subklassene PrefereceScreen og PreferenceCategory. Disse to brukes til å organisere verdiene i grupper. Vi starter alltid med en PreferenceScreen. Inni der har vi null eller flere PreferenceCategory-elementer. Inni en PreferenceGroup kan vi putte så mange Preference-objekter vi ønsker. Det betyr at vi kan putte Preference-verdier direkte inn i en PreferenceScreen eller i en PreferenceCategory. Eller vi kan putte en PreferenceScreen inni en PreferenceCategory – dette vil resultere i at vi får opp et nytt vindu når vi klikker på det.

Andre subklasser av Preference er DialogPreference, TwoStatePreference og RingtonePreference. De to første av disse er abstrakte klasser, mens RingtonePreference er ikke-abstrakt. DialogPreference har subklassene EditTextPreference, ListPreference og MultiListPreference, mens TwoStatePreference har subklassene CheckBoxPreference og SwitchPreference. Det er disse (de ikke-abstrakte) klassene vi bruker til å sette og hente de ulike verdiene som vi ønsker å lagre.

Nå brukeren har satt/endret minst en verdi, lagres denne/disse verdien(e) i en fil i Android sitt filsystem. For å se på den, gjør du følgende i Android Studio:

1. Sørg for å ha en kjørende emulator
2. Start Android Device Monitor (finnes under Tools/Android)
3. Velg fanen File Explorer og ekspander katalogen data/data
4. Finn den applikasjonen du vil se preferanseverdiene for, og ekspander den.
5. Ekspander katalogen shared_prefs og klikk på den fila som ligger der (hvis katalogen ikke finnes, er ingen preferanser blitt lagret ennå).
6. Klikk på knappen med lyserød venstrepil øverst til høyre i vinduet. Da får du opp et Filutforsker-vindu der du kan bestemme hvor fila skal lagres.
7. Nå kan du ta fram fila i Android Studio eller et annen passende tekstredigeringsverktøy.

Etter å ha kjørt lærebokas eksempel, fikk jeg følgende preferansefil:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="envId">test</string>
    <string name="account">mildrid</string>
    <boolean name="allowMultipleUsers" value="true" />
</map>
```

Ofte er det hensiktsmessig å lage Preference-objektene ved hjelp av XML. Læreboka viser et eksempel på dette i Listing 16.6. Her har de laget en skjerm med to kategorier. Inni den første kategorien har de puttet en CheckBoxPreference, mens de i den andre har puttet to EditTextPreference. Her er et eksempel på hvordan vi kan lage en ListPreference:

```
        <ListPreference
                android:title="List"
                android:entries="@array/PrefList"
```

```
android:entryValues="@array/PrefList"
 android:summary="Click and choose"
android:key="listPref"/>
```

Title og summary er det som vises fram på skjermen, mens key er den nøkkelen som verdien lagres med. Dette gjelder alle Preference-typene. entries og entryValues er derimot spesifikke for ListPreference (og MultiChoiceListPreference). entries er en tabell over de verdiene som skal vises fram i lista. Den har jeg har definert i strengtabellen PrefList. entryValues er hvilke verdier som skal lagres ved de ulike valgene. Jeg har latt det være de samme tekstene som de som vises fram, men det er fullt mulig å la det være noe annet.

Den XML-fila som er laget, må plasseres i en katalog som heter xml (som må lages) under katalog res. For å vise fram disse preferansene lager læreboka en ny aktivitet, SettingsAcivity. Inni der lages et fragment som bruker en subklasse av PreferenceFragment. I denne klassens onCreate() kalles metoden addPreferencesFromResource(R.xml.filnavn), dermed vises alle preferansene, og bruker kan endre en eller flere av dem.

Det er jo ingen vits i ha slike verdier hvis vi ikke kan bruke dem. Derfor må de kunne hentes fram slik at programmet kan teste på dem og tilpasse seg det brukeren har angitt. Det gjøres med metoden getDefaultSharedPreferences(). Metoden returnerer et objekt av typen SharedPreferences (et objekt av en klasse som implementerer grensesnittet SharedPreferences). På dette objektet kan vi kalle getString(nøkkel, defaultVerdi) hvis verdien er av strengtype (fra EditTextPreference eller ListPreference), eller getBoolean(nøkkel, defaultVerdi) hvis verdien er av type boolean (fra CheckBoxPreference eller SwichPreference). Defaultverdien brukes hvis nøkkelen ikke finnes i fila (bare ting som er endret finnes her, det holder ikke at preference-skjermen er vist fram, det må være foretatt endring på akkurat denne preferansen for at nøkkelen skal eksistere i fila). For å få en konsistent oppførsel på programmet bør defaultverdien her være den samme som den som eventuelt er angitt i XML-fila.

Det er mulig å endre preferansene uten å gå via preferanseskjermen. Det gjøres ved å hente fram preferanseobjektet (på samme måte som over) samt skaffe et objekt av typen Preferances.Editor. Så kan man bruke putString() og/eller putBoolean() på dette objektet. Til slutt må man ta commit(). Her er et eksempel:

```
SharedPreferences appPrefs =
      getDefaultSharedPreferences(this);
SharedPreferences.Editor prefsEditor = appPrefs.edit();
prefsEditor.putString(nøkkel1,tekst);
prefsEditor.putBoolean(nøkkel2,boolean_verdi);
prefsEditor.commit();
```

7.2 Lese fra og skrive til datafil

Hvis man har større mengder med data, er det antakelig best å bruke vanlige filer, ikke Preference-mekanismen. Man kan lagre både i det interne minnet og i et eksternt lager, for eksempel et SD-kort. Fordelen med det siste er både at det vanligvis er mer tilgjengelig plass der, og at kortet kan tas ut og settes inn i en annen mobil enhet hvis det er behov for å gi andre tilgang til fila. Bortsett fra måten man framskaffer fila på, gjøres lesing og skriving på samme måte i begge tilfellene. I det interne lageret er dataene private, så det er bare applikasjonen som har lagret dem som kan hente dem fram igjen, mens data lagret i det eksterne lagret kan hentes fram av andre applikasjoner og brukere. I læreboka er konstantene

MODE_WORLD_READABLE og MODE_WORLD_WRITEABLE nevnt, men de er «deprecated» og bør ikke brukes.

Hvis vi skal bruke det interne lageret, kan vi skrive:

```
File dir = getFilesDir();
File file = new File(dir, fileName);
PrintWriter writer = null;
try {
    writer = new PrintWriter(file);
    writer.write(body);
…osv
```

(for skriving)

eller

```
File dir = getFilesDir();
File file = new File(dir, fileName);
FileReader fileReader = null;
BufferedReader bufferedReader = null;
try {
    fileReader = new FileReader(file);
    bufferedReader = new BufferedReader(fileReader);
    String line = bufferedReader.readLine();
…osv
```

(for lesing)

For å finne igjen fila, eller plassere den på rett plass, kan du (i Android Studio) gjøre det samme som for å finne igjen preferansefila. Se etter katalogen files i data/data/pakkenavn. Hvis du skal legge inn en fil, må kanskje katalogen files opprettes, det kan du gjøre ved å trykke på den grønne +-en. Så kan fila lastes opp ved å trykke på den lyserøde høyre-pilen.

Hvis vi derimot skal bruke eksternlageret, kan vi skrive noe slikt som

```
File dir = getExternalFilesDir(null);
File file = new File(dir, fileName);
…osv
```

Hvis vi har en ferdig fil som skal leses inn, kan den alternativt legges i katalogen res/raw slik at den får en id som starter med R.raw og ender med filnavnet. Hvis fila for eksempel heter textfile.txt, vil id være R.raw.textfile. Da kan den leses slik:

```
private String readFile(int id) {
  StringBuffer sb= new StringBuffer("");
  try {
        InputStream is = getResources().openRawResource(id);
        BufferedReader reader= new BufferedReader(new InputStreamReader(is));
        String line = reader.readLine();
        while (line != null) {
              sb.append(line);
              sb.append("\n");
              line = reader.readLine();
        }
        reader.close();
  }
  catch (IOException e) {
        e.printStackTrace();
```

```
        }
        return sb.toString();
    }
```

7.3 Bruke database

Android har innebygd støtte for databasesystemet SQLite. Læreboka har et eksempel med en enkel database, men hvis du skal lage en database selv, bør du nok kunne noe mer om SQLite enn det som står i læreboka, se for eksempel [SQLite].

For å bruke databasen, trenger vi et objekt av klassen SQLiteDatabase samt et objekt av klassen SQLiteOpenHelper. I konstruktøren til SQLiteOpenHelper oppgis navnet på databasen, slik at det kan knyttes forbindelse mellom objektet og databasen. Så kan metoden getWriteableDatabase() brukes til å framskaffe selve databaseobjektet. Dette databaseobjektet kan vi så utføre SQL-kommandoer på, blant annet har vi metodene insert(), delete() og query() som, sammen med de medsendte argumentene, blir omformet til SQL-setninger og sendt til databasen. Når vi er ferdig med å bruke databasen, kan vi lukke den ved å bruke SQLiteOpenHelper-metoden close().

Det er en god del ting som må administeres i forbindelse med databasebruk, så det er lurt å la en egen klasse sørge for dette, slik at selve applikasjonen (Activity-klassen) bare trenger å forholde seg til enkle kommandoer for å få gjort det som skal gjøres. Læreboka gjør det på denne måten, og lager klassen DatabaseManager som en subklasse klasse fra SQLiteOpenHelper. Der definerer de for eksempel addContact() som sørger for å framskaffe databasen, sette inn den nye kontakten og lukke databasen igjen. På denne måten slipper Activity-klassen å tenke på alt som faktisk må gjøres for å sette inn en ny kontakt.

For å putte verdier inn i en rekke i en databasetabell, må vi først putte dem inn i et objekt av typen ContentValues, deretter kan vi sette inn dette objektet i databasen.

SQLiteOpenHelper er en abstrakt klasse, så man er nødt til å definere sin egen subklasse. De to metodene onCreate() og onUpdate() er abstrakte og må defineres i subklassen. I onCreate skal man sørge for å opprette databasen hvis den ikke eksiterer.

Når vi skal hente ut informasjon fra databasen, brukes metoden query(). Som returverdi fra den får vi et objekt av typen Cursor (nærmere bestemt klassen SQLiteCursor). Dette objektet kan brukes til å iterere gjennom de returnerte tabellrekkene. Noen av metodene i dette grensesnittet er getColoumnNames(), getCount(), getFloat(kolonnenr), getInt(kolonnenr), getString(kolonnenr), getType(kolonnenr), getPosition(), moveToFirst(), moveToLast(), moveToNext(), moveToPrevious(), move(strekning), moveToPosition(nyPosisjon), isFirst(), isLast(), isNull().

Nedenfor følger et eksempel. Det er basert på lærebokas eksempel, men er noe utvidet.

7.4 Eksempel på bruk av database

Vi skal lage en database med bøker. En bok har en tittel og en eller flere forfattere. I en virkelig applikasjon ville vi tatt med flere ting, men her nøyer vi oss med disse to attributtene. Siden en forfatter kan ha skrevet mer enn en bok og en bok kan være skrevet av flere forfattere, får vi en mange-til-mange-relasjon mellom tittel og forfatter, og trenger derfor tre tabeller: En tabell over forfattere, en for titler og en for koplingen mellom forfatter og tittel.

Vi følger lærebokas eksempel og lager klassene DatabaseManager og MainActivity. Først må vi formulerer de setninger som trengs for å opprette databasen. Det gjøres ved å lage SQL-setninger. Men for å gjøre det mest mulig fleksibelt, lager jeg aller først diverse navngitte konstanter:

```
static final String TABLE_AUTHOR = "author";
static final String TABLE_TITLE = "book";
static final String TABLE_AUTHOR_BOOK = "author_book";
static final String KEY_ROWID = "_id";
static final String KEY_NAME = "name";
static final String KEY_TITLE="title";
static final String KEY_AUTHOR="author_id";
static final String KEY_BOOK="book_id";
```

Så kommer sql-setningene:

```
static final String DATABASE_CREATE1 = "create table " + TABLE_AUTHOR
        + " ("+KEY_ROWID+" integer primary key autoincrement, "
        + KEY_NAME +" text unique not null);";
static final String DATABASE_CREATE2 = "create table " + TABLE_TITLE
        + " ("+KEY_ROWID+" integer primary key autoincrement, "
        + KEY_TITLE +" text unique not null);";
static final String DATABASE_CREATE3 = "create table " + TABLE_AUTHOR_BOOK
        + " ("+KEY_ROWID+" integer primary key autoincrement, "
        + KEY_BOOK+" numeric, " + KEY_AUTHOR+" numeric,"
        + "FOREIGN KEY("+KEY_AUTHOR+") REFERENCES " + TABLE_AUTHOR
        + "("+KEY_ROWID+"), "+ "FOREIGN KEY("+KEY_BOOK
        +") REFERENCES "+ TABLE_TITLE +"("+KEY_ROWID+")" + ");";
```

Første setning sier at tabellen author skal ha to kolonner, _id og name, der _id er primærnøkkel og økes automatisk for hver rekke. name er en tekststreng som ikke kan være null og samme tekst kan ikke finnes flere ganger. Andre setning lager tabellen book på samme måte. Tredje tabell kopler sammen en bok-id og en forfatter-id slik at vi kan holde rede på hvilken forfatter som har skrevet hvilken bok. Vi forteller at book_id skal være _id i tabellen book og tilsvarende for author_id.

Vi definerer også konstantene

```
static final String DATABASE_NAME="BooksDb";
static final int DATABASE_VERSION=1;
```

Siden konstruktøren har parameter Context, tar vi vare på denne i en objektvariabel, slik at vi har den for hånden om vi trenger den (f.eks. hvis vi vil lage en Toast til hjelp underveis i utviklingen av programmet).

```
private Context context;
public DatabaseManager(Context context) throws Exception {
        super(context,
                /*db name=*/ DATABASE_NAME,
                /*cursorFactory=*/ null,
                /*db version=*/DATABASE_VERSION);
        this.context = context;
}
```

Så går vi løs på metodene. Først de to som er påkrevd:

```
public void onCreate(SQLiteDatabase db) {
    Log.d("db", "onCreate");
    db.execSQL(DATABASE_CREATE1);
```

```
    db.execSQL(DATABASE_CREATE2);
    db.execSQL(DATABASE_CREATE3);
}

@Override
public void onUpgrade(SQLiteDatabase db, int arg1, int arg2) {
    Log.d("db", "onUpdate");
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_AUTHOR_BOOK);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_TITLE);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_AUTHOR);
            // re-create the table
    onCreate(db);
 }
```

onCreate() opprettes de tre tabellene, mens de i onUpgrade slettes og opprettes på nytt. Begge disse får inn databasen som parameter, slik at vi slipper å hente det fram selv.

Vi lager også en metode clean() som renser tabellene for innhold. Den er kjekk å ha under utprøving av programmet for å kunne starte med en tom database. Den ser slik ut:

```
public void clean() {
    SQLiteDatabase db = this.getWritableDatabase();
    onUpgrade(db, 0, 0);
    db.close();
}
```

Så kan vi gå løs på de metodene som setter inn data i databasen. Vi lager de private metodene insertAuthor(), insertBook() og insertAuthorBook() samt den offentlige insert(). De tre første setter inn i hver sin tabell, og er hjelpemetoder til den offentlige.

```
private long insertAuthor(String name, SQLiteDatabase db)  {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    long id = db.insert(TABLE_AUTHOR, null, initialValues);
    return id;
}
private long insertBook(String title, SQLiteDatabase db) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_TITLE, title);
    long id = db.insert(TABLE_TITLE,  null, initialValues);
    return id;
}
private long insertAuthor_Book(long author, long book, SQLiteDatabase db) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_AUTHOR, author);
    initialValues.put(KEY_BOOK, book);
    long id = db.insert(TABLE_AUTHOR_BOOK,  null, initialValues);
    return id;
}
```

For å sette inn en forfatter, framskaffer vi altså et objekt av klassen ContentValues, putter inn forfatternavnet og setter det inn i forfattertabellen.

Metoden InsertBook lages på helt tilsvarende måte og insertAuthor_Book bare litt annerledes.

Alle tre metodene returnerer nummeret på den rekken datane ble satt inn i. Første rekke er nummer 1, ikke 0.

Så kommer den metoden som skal brukes i Activity-klassen:

```
public long insert(String author, String title){
    SQLiteDatabase db = null;
    db = this.getWritableDatabase();
    Cursor cursor =
        db.query(true, TABLE_AUTHOR, new String[]{KEY_ROWID, KEY_NAME},
        KEY_NAME + "='" + author + "'", null, null, null, null, null);
    long author_id;
    if (cursor == null || cursor.getCount() == 0) {
        author_id = insertAuthor(author,db);
    } else {
        cursor.moveToFirst();
        author_id = cursor.getLong(0);
    }
    long title_id;
    cursor =
        db.query(TABLE_TITLE, new String[]{KEY_ROWID, KEY_TITLE},
        KEY_TITLE + "='" + title + "'", null, null, null, null, null);

    if (cursor == null || cursor.getCount() == 0) {
        title_id = insertBook(title,db);
    } else {
        cursor.moveToFirst();
        title_id = cursor.getLong(0);
    }
    long id = insertAuthor_Book(author_id, title_id,db);
    db.close();
    return id;
}
```

Her ber vi først om å få ut data om den gitte forfatteren. Så sjekker vi om forfatteren ble funnet ved å sjekke antallet rekker som ble returnert. Hvis antallet er 0, betyr det at forfatteren ikke fantes, så da setter vi ham/henne inn. Hvis antallet er større enn 0, henter vi fram dens _id ved å kalle getLong(0) siden _id ligger i kolonne 0. Samme operasjon blir gjort med tittelen, og når vi har framskaffet både forfatter-id og tittel-id, kan vi sette dem inn i forfatter-bok-tabellen.

For å hente fram alle forfatterne, lager vi følgende metode:

```
public ArrayList<String> getAllAuthors()  {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor =
            db.query(TABLE_AUTHOR, new String[]{KEY_ROWID, KEY_NAME}, null,
null, null, null, null, null);
    ArrayList<String> res = new ArrayList<>();
    if (cursor != null) {
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            res.add(cursor.getString(1));
            cursor.moveToNext();
        }
    }
    db.close();
    return res;
}
```

Vi forteller hvilken tabell vi ønsker å lete i (TABLE_AUTHOR) og hvilke kolonner vi vil ha tak i (_id og name). Tilbake får vi en cursor som vi kan hente ut kolonner i. getLong(0) gir id mens getString(1) gir navnet. Så kan vi gå til neste rekke ved å bruke moveNext().Vi kan hente alle navnene ved å lage løkke som går så lenge vi ikke har kommet forbi slutten.

Når det er mer kopliserte SQL-setninger som som skal utføres, kan vi bruke

```
public Cursor rawQuery (String sql, String[] selectionArgs)
```

Som eksempel på dette, skal vi finne alle bøkene en bestemt forfatter har laget. Først lager vi SQL_setningen:

```
static final String AUTHOR_BOOK_SELECTION =
        "select "+ TABLE_TITLE+"."+KEY_ROWID+", "+TABLE_TITLE+"."+KEY_TITLE
      + " from " + TABLE_AUTHOR+", "+TABLE_TITLE + ", "+TABLE_AUTHOR_BOOK
      + " where "+ TABLE_AUTHOR + "."+KEY_ROWID+"="+ TABLE_AUTHOR_BOOK
      + "."+KEY_AUTHOR+" and "+ TABLE_TITLE + "."+KEY_ROWID+"="
      + TABLE_AUTHOR_BOOK + "."+KEY_BOOK+" and "+ TABLE_AUTHOR +"."
      + KEY_NAME +"=?";
```

Vi velger altså id og tittel fra bok-tabellen, og krever at både forfatter-id og bok-id skal stemme med det vi finner i koplingstabellen author_book, mens forfatterens navn skal være noe som foreløpig er ukjent.

Så lager vi metoden

```
public ArrayList<String> getBooksByAuthor(String author)  {
    SQLiteDatabase db = this.getReadableDatabase();
    ArrayList<String> res = new ArrayList<>();
    Cursor cursor =
            db.rawQuery(AUTHOR_BOOK_SELECTION, new String[]{author});
    if (cursor != null) {
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            res.add(cursor.getString(1));
            cursor.moveToNext();
        }
     }
    db.close();
    return res;
}
```

Forfatternavnet setter vi altså inn i en streng-tabell, og det fungerer slik at det første elementet i tabellen puttes inn i første ?. Har vi flere ? i setningen vår, fortsettes utover i strengtabellen (men det har vi ikke her). Det er også mulig å bruke ? på samme måte i den vanlige query-metoden.

For å teste ut min database, lager jeg en Activity med følgende onCreate()-metode:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    getActionBar().setDisplayHomeAsUpEnabled(true);
    try {
        db = new DatabaseManager(this);
         db.clean();
         long id = db.insert("Bud Kurniawan","Android Application
Development: A Beginners Tutorioal");
        id = db.insert("Mildrid Ljosland", "Programmering i C++");
        id = db.insert("Else Lervik", "Programmering i C++");
```

```
        id = db.insert("Mildrid Ljosland", "Algoritmer og datastrukturer");
        id = db.insert("Helge Hafting", "Algoritmer og datastrukturer");
        ArrayList<String> res = db.getAllAuthors();
     //   ArrayList<String> res = db.getBooksByAuthor("Mildrid Ljosland");

        showResults(res);
    }
     catch (Exception e) {
        String tekst = e.getMessage();
        TextView t = (TextView)findViewById(R.id.tw1);
        t.setText(tekst);
     }
}
```

showResults() er en enkel metode som viser fram resultatet i en TextView:

```
public void showResults(ArrayList<String> list) {
    StringBuffer res = new StringBuffer("");
    for (String s : list)  {
        res.append(s+"\n");
    }
    TextView t = (TextView)findViewById(R.id.tw1);
    t.setText(res);
}
```

Vedlagt denne leksjonen finner du hele prosjektet.

7.5 Referanser

[PullParser] http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html

[SQLite] http://www.sqlite.org/