Mobila tjänster och trådlösa nät HT 2012

HI1033

Lecturer: Anders Lindström,
anders.lindstrom@sth.kth.se

Lecture 6
Today's topics

- Files
- Android Databases - SQLite
- Content Providers

Data storage

- Resources (res/raw)
  - private data installed with the application (read-only)
- Shared PreferencesStore
  - private primitive application data in key-value pairs
- Internal StorageStore
  - private data on the device memory
- External StorageStore
  - public data on the shared external storage
- SQLite Databases
  - structured data in a private database
- Network Connection
  - data on the web; read write to server (e.g. a DB)

Files, internal storage

- By default, files saved to the internal storage are private to the application (in the current applications folder)
- When the user uninstalls the application, these files are removed ☺
- FileOutputStream fos = openFileOutput(
  fileName, Context.MODE_PRIVATE);
  fos.write(string.getBytes());
  fos.close();

Files, internal storage

- MODE_PRIVATE, MODE_APPEND, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE
- getFilesDir() Gets the absolute path to the filesystem directory where your internal files are saved
- getDir() Creates (or opens an existing) directory within your internal storage space
- deleteFile() Deletes a file saved on the internal storage
- fileList() Returns an array of file names

Files, internal storage

- getFileInput/getFileOutput returns a primitive stream object
- Use the classes in java.io to add filter or wrapper streams
- Example, writing to a text file:

```
PrintWriter pw = new PrintWriter(
        getFileOutput(. . .));
pw.println(info);
```

Writing to a text file

```
PrintWriter writer = null;
try {
        OutputStream os = this.openFileOutput(
                fileName, Context.MODE_PRIVATE);
        writer = new PrintWriter(os);

        writer.println("Lots of information to remember");
        . . .
}
catch(IOException ioe) {
   showToast("Error while writing to file");
}
finally {
   if(writer != null) writer.close();
}
```

Reading from a text file

```
BufferedReader reader = null;
try {
    InputStream is = this.openFileInput(fileName);
    reader = new BufferedReader(new InputStreamReader(is));

    String line = reader.readLine();
    while(line != null) {
        . . . // Process the information
        line = reader.readLine();
    }
}
catch(IOException ioe) {
    showToast("Error while writing to file");
}
finally {
    try {
        if(reader != null) reader.close();
    }
    catch(IOException ioe) {}
}
```

Files, external storage

- Android devices support a shared "external storage"
- Removable storage media (such as an SD card) or an internal (non-removable) storage
- Files saved to the external storage are world-readable
- Files can be modified by the user when they enable USB mass storage to transfer files on a computer

Files, external storage

- Call getExternalStorageState() to check whether the media is available
- API Level 7 or lower
  - getExternalStorageDirectory(), opens a java.io.File representing the root of the external storage
  - Write your data in /android/data/<*package_name*>/files/
- API Level 8 or greater
  - getExternalFilesDir() opens a java.io.File representing the root of the external storage for your application

Reading “static” files from resources

- External files resources can be included as resources in the res/*raw* directory in your project
- Read-only!
- Resources res = context.getRecources();
  InputsStream is =
  res.openRawResource(R.raw.filename);

Reading from resources

```
BufferedReader reader = null;
try {
    Resources res = this.getResources();
    InputStream is = res.openRawResource(R.raw.myfile);
    reader = new BufferedReader(new InputStreamReader(is));

    String line = reader.readLine();
    while(line != null) {
        textOutput.append(line + "\n");
        line = reader.readLine();
    }
}
finally { . . .
```

Files, readings

- http://developer.android.com/guide/topics/data/data-storage.html
- Meier, chapter 6

Databases

- Organize, store, and retrieve (large amounts of) *structured* data easily
- Database management systems, DBMS,
  - create databases (including tables and such)
  - allow data creation and maintenance
  - search for data and other access
- Structured Query Language, SQL, language designed for managing data in relational database management systems

Databases

- Atomicity - modifications must follow an "all or nothing" rule
- Consistency - only valid data will be written to the database
- Isolation - operations cannot access data that has been modified during a transaction that has not yet completed
- Durability - once the user has been notified of a transaction's success the transaction will not be lost

SQLite

- SQLite
  - Lightweight
  - Reliable
  - Standards compliant
  - Open-source
- A SQLite database is an integrated part of the application that created it
  - reducing external dependencies
  - simplifies transaction locking and synchronization
- Used in iPhone, several MP3-players, Adobe, Firefox, Airbus, …

Android Databases

- By default private to the application
- Stored in directory
  /data/data/<package name>/databases
  on the device
- Used in native applications such as the contact manager and media store
- Design consideration(s)
  – Store files (bitmaps, audio, …) outside the DB. Store a URI in DB

Open or create a database

- ```
  database = context.openOrCreateDatabase(
    "dictionary.db",
     CONTEXT.MODE_PRIVATE,
     null);
  database.setLocale(Locale.getDefault());
  database.setLockingEnabled(false);
  database.setVersion(1);
  ```

- LockingEnabled: SQLiteDatabase is made thread-safe by using locks around critical sections
- LockingEnabled is expensive
  - If your DB will only be used by a single thread then you should set this to false
  ```

Execute SQL statements

- execSQL(…), executes a single statement that is *not a query*
- insert(…), update(…), delete(…)
- query(…)
- SQLiteQueryBuilder, more complex queries
- Queries return a *Cursor* object – a reference to the data

Create a table

- ```
  String DATABASE_CREATE =
  "CREATE TABLE table_word
  (_id INTEGER PRIMARY KEY AUTOINCREMENT, word
  TEXT NOT NULL, definition TEXT NOT NULL);";
  ```

- ```
  database.execSQL(DATABASE_CREATE);
  ```

- startManagingCursor requires row id to be named “_id” !

Insert

- Use ContentValues to provide column names and column values
- ```
  ContentValues vals = new ContentValues();
  vals.put("word", word);
  vals.put("definition", definition);
  ```
- ```
  long id = database.insert("table_word", null,
  vals);
  ```
- insert() returns the row id or -1
- Alternative:
  ```
  String ins =
  "INSERT INTO table_word (word, definition) VALUES
  ('" + word + "','" + definition + "')";
  execSQL(ins);
  ```

Update

- Use ContentValues to provide column names and column values
- ```
  ContentValues args = new ContentValues();
  args.put(KEY_WORD, word);
  args.put(KEY_DEFINITION, definition);
  ```
- ```
  int n = database.update(
      TABLE_WORDS, args, KEY_ROWID + "="     +
  rowId, null);
  ```
- Returns the number of rows updated

Delete

- delete (String table, String whereClause, String[] whereArgs)

- ```
  int n = database.delete(TABLE_WORDS,
      KEY_ROWID + "=" + rowId, null);
  ```

- Returns the number of rows affected if a whereClause is provided, 0 otherwise
  ```

Transactions

- Multiple operations that should happen all together, or not at all
- *database.beginTransaction();*

```
try {
        // insert/delete/update records
        // …
        database.setTransactionSuccessful();
}
finally {
        database.endTransaction();
}
```

- setTransactionSuccessful + endTransaction *commits* the changes
- endTransaction without setTransactionSuccessful causes a *roll back* on all changes

Queries, Cursor

- Query results are accessed using a Cursor, allowing random access to the query result

- ```
  // Query: SELECT * FROM table_words
  Cursor c = database.query(TABLE_WORDS, null,
  …);
  // Do something (quick) with cursor…
  c.close();
  ```

- Cursor methods
  - moveToFirst/Next/Previous
  - getCount (rows)
  - getColumnName/Names
  - moveToPosition/getPosition
  ```

Queries, Cursor

```
Cursor cursor = database.query(…);
int cols = cursor.getColumnCount();
cursor.moveToFirst();
while(cursor.isAfterlast() == false) {
    String rowResult = “”;
    for(int i = 0; i < cols; i++) {
            rowResult += cursor.getString(i) + “, “;
    }
    cursor.moveToNext();
}
cursor.close();
```

Queries, Cursor

- Longer cursor tasks, manage cursor as part of application lifecycle
  - on pause, deactivate cursor
  - on resume, refresh cursor
  - on destroy, close cursor
- Alternative: call activity.startManagingCursor(cursor)
  - the Activity will handle the life cycle calls for the cursor
- startManagingCursor requires key to be named “_id”

Queries

- query(String table, String[] columns,
  String selection, String[] selectionArgs,
  String groupBy, String having, String orderBy)
- SELECT _id, word, definition FROM table_words
  translates to…
- ```
  Cursor cursor =
      database.query(
            TABLE_WORDS,
            new String[] { KEY_ROWID,
                  KEY_WORD, KEY_DEFINITION },
            null, null, null, null, null
      );
  ```
  ```

SimpleCursorAdapter

- Adapter to map columns from a cursor to TextViews or ImageViews defined in an *XML layout file*

```
startManagingCursor(cursor);
cursor.moveToFirst();
String[] from = new String[] {KEY_WORD, KEY_DEFINITION };
int[] to =
       new int[] { R.id.WordView, R.id.DefinitionView };
SimpleCursorAdapter entries =
       new SimpleCursorAdapter(this,
              R.layout.entry_row, cursor, from, to);
resultView.setAdapter(entries);
```

SQLiteOpenHelper

- A helper class to manage database creation, version management, … - i.e. the database's life cycle

```
private class DatabaseHelper extends SQLiteOpenHelper {

  DatabaseHelper(Context context) {
       super(context, DATABASE_NAME, null,
       DATABASE_VERSION);
  }

  public void onCreate(SQLiteDatabase db) {
       db.execSQL(DATABASE_CREATE);
  }

  public void onUpgrade(SQLiteDatabase db,
              int oldVersion, int newVersion) {
       // Manage database upgrade
  }
  . . .
```

(Android) Databases, design considerations

- Use SQLiteOpenHelper to manage database creation and version management
- Write an “adapter class”, with (strongly typed) methods, hiding the database manipulation, and constants repr keys, …
- Example DictionaryDbAdapter.java

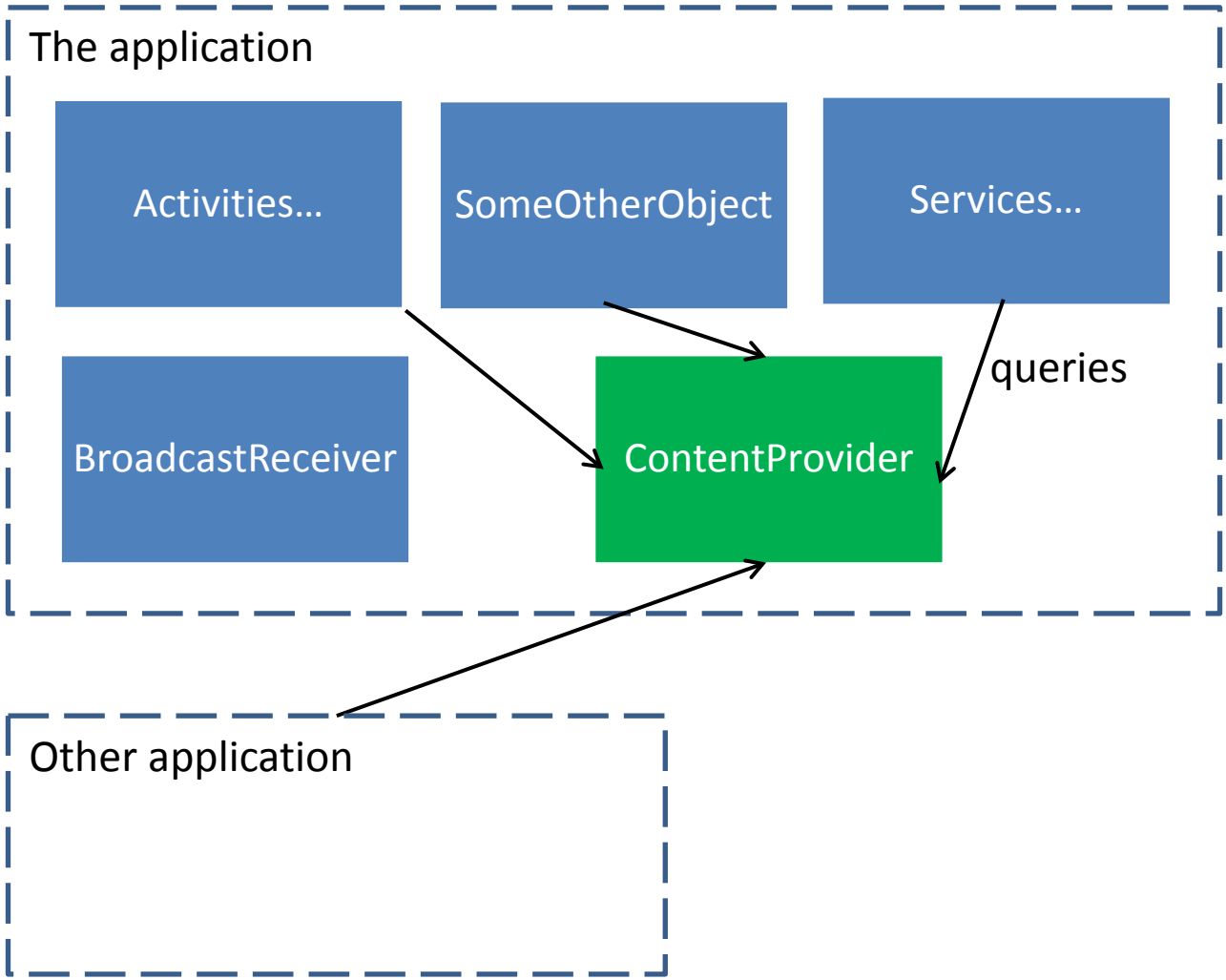
```
public Cursor fetchMatchingWords(String str){
    Cursor cursor = database.query(…);
    return cursor;
}
```

- Model rows as class instances
- SQLite does not enforce foreign key constraints – use triggers instead (via execSQL)
- Don’t store files (media and others) in the database

Links

- SQL syntax, e.g.: http://www.w3schools.com/sql/default.asp
- SQLite: http://www.sqlite.org/
- …and of course developers.android.com
- sqlite3 – command line tool to monitor/ interact with the database during development/testing

Content Providers



Content Providers

- Content providers store and retrieve data, hiding the underlying data source
- Shared Content Providers makes it possible to query and update other applications records (if permission granted)

- The only way to share database content across applications

Content Providers

- Each content provider exposes a public URI (wrapped as a Uri object) that uniquely identifies its data set
- Begins with "content:// "
- Custom ContentProvider Uri:
  ```
  public static final CONTENT_URI = Uri.parse(
  ”content://se.kth.anderslm.MyCustomProvider”);
  ```

- Android defines CONTENT_URI constants for all native providers, e.g. android.provider.Contacts.Photos.CONTENT_URI

Content Providers

- Abstract class ContentProvider
  - onCreate(), called to initialize the provider
  - query(), returns a Cursor
  - insert()
  - update()
  - delete(), delete data from the provider
  - getType() , returns the MIME type of the data
  - …

- Some native content providers
  MediaStore, CallLog, Browser, Contacts, UserDictionary, …

Querying a Content Provider

- ContentProvider acts as an interface that clients use *indirectly*, through ContentResolver objects or Activities

- ```
  ContentResolver cr =
      activity.getContentResolver();
  Cursor c = ContentResolver.query(…);
  ```

- ```
  Cursor c = activity.managedQuery(…);
  ```

Querying a Content Provider

- Retrieve a list of contact names and their primary phone numbers

```
import android.provider.Contacts.People;
// Form an array specifying which columns to return
String[] projection = new String[] {
                                People._ID,
                                People._COUNT,
                                People.NAME,
                                People.NUMBER
                            };
// Get the base URI of the People table in the CP
Uri contacts =  People.CONTENT_URI;
```

Querying a Content Provider

- Retrieve a list of contact names and their primary phone numbers, continued

```
// Make the query
Cursor managedCursor = managedQuery(contacts,
       projection, // Which columns to return
       null,       // Which rows to return (all rows)
       null,       // Selection arguments (none)
       People.NAME + " ASC");  // Ascending order by name
```

Querying a Content Provider

- Reading names and phone numbers using the returned Cursor

```
if (cur.moveToFirst()) {

        String name, phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        . . .
        do {
                // Get the field values
                name = cur.getString(nameColumn);
                phoneNumber = cur.getString(phoneColumn);

                // Do something with the values
                ...
        } while (cur.moveToNext());
}
```

Create a custom Content Provider

- Extend abstract class ContentProvider, and override
  - onCreate(), called to initialize the provider
  - query(Uri, String[], String, String[], String), returns a Cursor
  - insert(Uri, ContentValues)
  - update(Uri, ContentValues, String, String[])
  - delete(Uri, String, String[])
  - getType(Uri) , returns the MIME type of the data
  - …

Create a custom Content Provider

- Add
```
public static final CONTENT_URI = Uri.parse(
”content://se.kth.anderslm.app.MyCustomProvider”);
```
- Add public static String constants to specify the columns
- Include an integer column "_id", with the constant _ID, for the IDs of the records
- Declare a <provider> element in the application's manifest file:
```
<provider
android:name=”se.kth.anderslm.app.MyCustomProvider”
android:authorities=”se.kth.anderslm.app.myproviderauth”
. . . />
```

Create a custom Content Provider

- Implementation, example

```
public class MyCustomProvider extends ContentProvider {
   // . . .
   public Cursor query(Uri uri, String[] projection, String selection,
         String[] selectionArgs, String sortOrder) {
      // . . .

      SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
      builder.setTables(DictionaryDatabase.TABLE_WORDS);
      SQLiteDatabase database = dbHelper.getReadableDatabase();

      Cursor cursor = builder.query(database, projection, selection,
          selectionArgs, null, null, sortOrder, null);
      cursor.setNotificationUri(getContext().getContentResolver(), uri);
      return cursor;
   }
   // . . .
}
```

Content Providers

- How to use existing Content Providers and create new ones: http://developer.android.com/guide/topics/providers/content-providers.html