Mobile Application Development – Android

Lecture 3

MTAT.03.262

Satish Srirama

satish.srirama@ut.ee

Mobile Cloud Lab

tiigriülikool

Programm IKT toetuseks kõrghariduses

Tiigriülikool on riiklik info- ja kommunikatsioonitehnoloogia-alane kõrghariduse toetusprogramm

Android Lecture 2 - recap

- Views and Layouts
- Events
- Basic application components
  - Activities
  - Intents

Outline

- Remaining basic application components
- Storage of data with Android
- Working with threads
- Home Assignment 1

Intents

- Explicit intent
- Implicit intent

BroadcastReceivers

- Used for system level message-passing mechanism
  - Components designed to respond to broadcast Intents
  - Allow you to register for system or application events
  - All registered *receivers* for an event will be notified by the Android runtime once this event happens
  - Example: applications can register for the ACTION_BOOT_COMPLETED system event
    - Fired once the Android system has completed the boot process
- Think of them as a way to respond to external notifications or alarms

Using BroadcastReceivers

- Example: Logging the phone number of calls

```
public class MyPhoneReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extras = intent.getExtras();
    if (extras != null) {
      String state = extras.getString(TelephonyManager.EXTRA_STATE);
      Log.w("MY_DEBUG_TAG", state);
      if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
        String phoneNumber = extras
            .getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
        Log.w("MY_DEBUG_TAG", phoneNumber);
      }
    }
  }
}
```

Using BroadcastReceivers - continued

- Register the BroadcastReceiver in manifest

```
<receiver android:name="MyPhoneReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" >
        </action>
    </intent-filter>
</receiver>
```

- You can also register a *broadcast receiver* dynamically via the `Context.registerReceiver()`
- You can also create custom intent and broadcast it with `sendBroadcast(intent);`

Exercise

- Receive a phone call and log the phone number

Content Providers

- Content providers manage access to a structured set of data
- Enable sharing of data across applications
  - Examples: address book, photo gallery, etc.
- Provides uniform APIs for:
  - querying (returns a Cursor)
  - delete, update, and insert rows
- Content is represented by URI and MIME type

Storage of data with Android

- We can put data into a preferences file.
- We can put data into a 'normal' file.
- We can use a local database on the handset
  - We can also use SQLite db
- We can send data across the network to a service

http://developer.android.com/guide/topics/data/data-storage.html

Preference files

- They are a light-weight option
- To save small collection of key-values
- Call `Context.getSharedPreferences()` to read and write values as key-value pairs
  - Use this if you need multiple preferences files identified by name
- Use `Activity.getPreferences()` with no name to keep them private to the calling activity
  - One preference file per activity and hence no name

Preference files - continued

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

```
// We need an Editor object to make preference changes.
// All objects are from android.context.Context
SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("silentMode", mSilentMode);
```

```
boolean silent = settings.getBoolean("silentMode", false);
```

- These are not sharable across applications
  - you can expose them as a ‘content provider’
- Used to store the state of an application

Files in Android (Internal storage)

- We can write larger data to file
- You can only access files available to the application
- Reading data from a file
  - `Context.openFileInput()` – Returns `FileInputStream` object
- Writing to a file
  - `Context.openFileOutput()` - Returns a `FileOutputStream` object
- If you want to save a static file in your application at compile time
  - `res/raw/mydata`
  - `You can open it with openRawResource(), passing the R.raw.<filename> resource ID`

Internal storage - continued

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

- Modes of access
  - MODE_PRIVATE - No access for other applications
  - MODE_WORLD_READABLE - Read access for other applications
  - MODE_WORLD_WRITABLE - Write access for other applications
- Accessing a shared file
  - FileInputStream openFileInput = createPackageContext("the_package", 0). openFileInput("thefile");

Exercise

- Working with files
  - Try to write a string to a file
  - Then read it back
  - Verify they are the same

External storage in Android

- Can access an external storage system e.g. the SD card
- All files and directories on the external storage system are readable for all applications with the correct permission
  - To read from external storage the application need to have the android.permission.READ_EXTERNAL_STORAGE permission
  - To write to the external storage it needs the android.permission.WRITE_EXTERNAL_STORAGE permission
- You get the path to the external storage system via the Environment.getExternalStorageDirectory() method

```
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    //  to know is we can neither read nor write
}
```

Internal vs External Storage

- Internal storage
  - It's always available
  - Files saved here are accessible by only your app by default
  - When the user uninstalls your app, the system removes all your app's files from internal storage
  - Internal storage is best when you want to be sure that neither the user nor other apps can access your files
- External storage
  - It is world-readable, so files saved here may be read outside of your control
  - External storage is the best place for files
    - that don't require access restrictions
    - that are to be shared with other apps
    - allow the user to access with a computer

Persisting data to a db

- Android API uses the built-in SQLite db
- SQLite is Simple, small (~350KB), light weight RDBMS implementation with simple API
- Each db is private to the application
  - You can expose the db as a content provider
- All databases, SQLite and others, are stored on the device in `/data/data/package_name/databases`

http://developer.android.com/training/basics/data-storage/databases.html#WriteDbRow

Creating SQL Databases

- Define a Schema and Contract
- Schema is a formal declaration of how the database is organized
- Create a companion class, *contract* class
  - A contract class is a container for constants that define names for URIs, tables, and columns
  - allows you to use the same constants across all the other classes in the package
  - So you change a column name in one place and have it propagate throughout your code

```
public final class FeedReaderContract {
    // To prevent someone from accidentally instantiating the contract class,
    // give it an empty constructor.
    public FeedReaderContract() {}

    /* Inner class that defines the table contents */
    public static abstract class FeedEntry implements BaseColumns {
        public static final String TABLE_NAME = "entry";
        public static final String COLUMN_NAME_ENTRY_ID = "entryid";
        public static final String COLUMN_NAME_TITLE = "title";
        public static final String COLUMN_NAME_SUBTITLE = "subtitle";
```

Persisting data to a db - continued

- To create a new SQLite database create a subclass of `SQLiteOpenHelper` and override the `onCreate()` method

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {
    // If you change the database schema, you must increment the database version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

```
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE + COMMA_S
    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP
    ... // Any other options for the CREATE command
    " )";
```

Persisting data to a db - continued

- `SQLiteDatabase` allows methods to open the database connection, perform queries and query updates, and close the database [`insert() update() and delete()`]
- `query()` and `rawQuery()`, both return a `Cursor` object

Put Information into a Database

- To access your database, instantiate your subclass `FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());`
- Insert data into the database by passing a `ContentValues` object to the `insert()` method

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);

// Insert the new row, returning the primary key value of the new row
long newRowId;
newRowId = db.insert(
         FeedEntry.TABLE_NAME,
         FeedEntry.COLUMN_NAME_NULLABLE,
         values);
```

Read Information from a Database

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    FeedEntry._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_UPDATED,
    ...
    };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(
    FeedEntry.TABLE_NAME,  // The table to query
    projection,                               // The columns to return
    selection,                                // The columns for the WHERE clause
    selectionArgs,                            // The values for the WHERE clause
    null,                                     // don't group the rows
    null,                                     // don't filter by row groups
    sortOrder                                 // The sort order
    );
```

Content Provider Basics

- All content providers implement a common interface for querying the provider and returning results
  - Also support adding, altering, and deleting data

```
public class ExampleProvider extends ContentProvider {
...
    // Creates a UriMatcher object.
    private static final UriMatcher sUriMatcher;
```

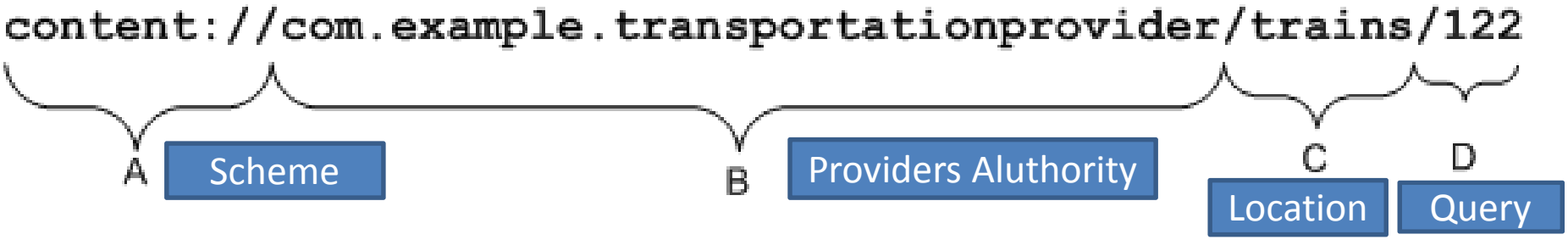
  - For creating content providers
    - http://developer.android.com/guide/topics/providers/content-provider-creating.html

Access Content Providers

- `ContentResolver` object from the application context provides access to the content provider
  - `ContentResolver cr = getContentResolver();`
- Content providers expose their data as a simple table on a database model
  - Each row is a record and each column is data of a particular type and meaning
  - Every record includes a numeric `_ID` field that uniquely identifies the record within the table
- The ContentResolver methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage

URIs of Content Providers

- Each content provider exposes a public URI
- A content provider that controls multiple tables exposes a separate URI for each one
- Example:



-
```
<provider android:name=".TransportationProvider"
        android:authorities="com.example.transportationprovider"
           . . .  >
```

- Until Android version 4.2 a content provider is by default available to other Android applications
  - From Android 4.2 a content provider must be explicitly exported android:exported=false|true

http://developer.android.com/guide/topics/providers/content-providers.html

Content Provider - example

Table 1: Sample user dictionary table.

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

- Words that might not be found in a standard dictionary
  - `content://user_dictionary/words`
- `Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);`

http://developer.android.com/guide/topics/providers/content-provider-basics.html

Querying a Content Provider

- To query a content provider you need
  - The URI that identifies the provider
  - The names of the data fields you want to receive
  - The data types for those fields
- The querying returns a `Cursor` object
- You can query either way
  - `ContentResolver.query()` or `Activity.managedQuery()`
  - Second one is better as it causes the activity to manage the life cycle of the Cursor until Android 3.0
- As of Android 3.0 `Activity.managedQuery()`is deprecated and you should use the `Loader` framework to access the ContentProvider
  - Should access ContentProviders asynchronously on a separate thread

Querying a Content Provider - continued

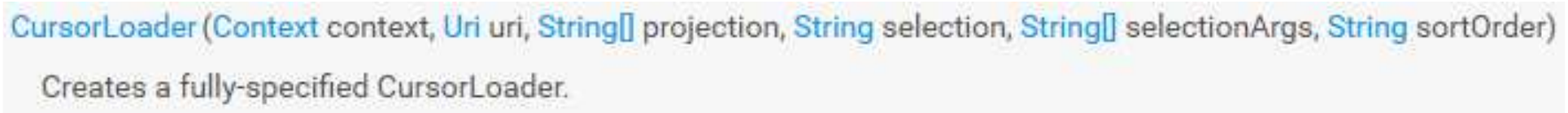
- Make the query

```
// A "projection" defines the columns that will be returned for each row
String[] mProjection =
{
    UserDictionary.Words._ID,    // Contract class constant for the _ID column name
    UserDictionary.Words.WORD,   // Contract class constant for the word column name
    UserDictionary.Words.LOCALE  // Contract class constant for the locale column name
};
```

```
// Does a query against the table and returns a Cursor object
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,  // The content URI of the words table
    mProjection,                       // The columns to return for each row
    mSelectionClause                   // Either null, or the word the user entered
    mSelectionArgs,                    // Either empty, or the string the user entered
    mSortOrder);                       // The sort order for the returned rows
```

Loaders

- They are available to every Activity and Fragment
- They provide asynchronous loading of data
- They monitor the source of their data and deliver new results when the content changes



http://developer.android.com/guide/components/loaders.html

Reading retrieved data

- Since a Cursor is a "list" of rows, a good way to display the contents of a Cursor is to link it to a `ListView` via a `SimpleCursorAdapter`.

```
// Defines a list of columns to retrieve from the Cursor and load into an output row
String[] mWordListColumns =
{
    UserDictionary.Words.WORD,   // Contract class constant containing the word column name
    UserDictionary.Words.LOCALE  // Contract class constant containing the locale column name
};

// Defines a list of View IDs that will receive the Cursor columns for each row
int[] mWordListItems = { R.id.dictWord, R.id.locale};

// Creates a new SimpleCursorAdapter
mCursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(),               // The application's Context object
    R.layout.wordlistrow,                  // A layout in XML for one row in the ListView
    mCursor,                               // The result from the query
    mWordListColumns,                      // A string array of column names in the cursor
    mWordListItems,                        // An integer array of view IDs in the row layout
    0);                                    // Flags (usually none are needed)

// Sets the adapter for the ListView
mWordList.setAdapter(mCursorAdapter);
```

Adapters

- Sometimes you may want to bind your view to an external source of data
  - Example: A string array or list extracted from DB
- View is initialized and populated with data from an Adapter
- Example:

```
//Get references to the UI widgets
ListView myListView = (ListView)findViewById(R.id.myListView);
//create the array list of to do items
final ArrayList<String> todoItems = new ArrayList<String>();
final ArrayAdapter<String> aa;
aa = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,todoItems);
// Bind the array adapter to the list view
myListView.setAdapter(aa);
todoItems.add(0,"satish");
aa.notifyDataSetChanged();
```

http://www.vogella.com/articles/AndroidListView/article.html

Content Provider – Example
Reading contact names and phone nos

```
private Cursor getContacts() {

    Uri uri = ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
    String[] projection    = new String[] {
            ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME,
            ContactsContract.CommonDataKinds.Phone.NUMBER};

    Cursor people = getContentResolver().query(uri, projection, null, null, null);

    return people;
}

Cursor people = getContacts();

int indexName = people.getColumnIndex(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME);
int indexNumber = people.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER);

people.moveToFirst();
do {
    String name   = people.getString(indexName);
    String number = people.getString(indexNumber);
    // Do work...

} while (people.moveToNext());
```

Exercise

- Display the contact names and phone numbers

- The contacts API is extremely tricky and has several implicit joins
  - Read it as per your interest

http://developer.android.com/guide/topics/providers/contacts-provider.html

Services

- Faceless components that run in the background
  - Example: music player, network download, etc.
- Can run in your own process or separate process
- They can perform long-running operations in the background
  - They have higher priority than the background activities
    - So safe from the runtime memory management
- A service can essentially take two forms
  - Started - startService() - run in the background indefinitely, even if the component that started it is destroyed
  - Bound – An application component binds to the service by calling bindService()

Services - continued

- Explicitly starting new Service

```
Intent intent = new Intent(this,
  HelloService.class);
  startService(intent);
```

- Services also have their life cycles managed
- You can also start java threads in Services

http://developer.android.com/guide/topics/fundamentals/services.html

Homework

- Start a service to play music in the background

Process Management in Android - recap

- By default in Android, every component of a single application runs in the same process
- When the system wants to run a new component:
  - If the application has no running component yet, the system will start a new process with a single thread of execution in it
  - Otherwise, the component is started within that process
- If you want a component of your application to run in its own process, you can still do it through the `android:process` XML attribute in the manifest
- The system might decide to kill a process to get some resources back
  - Priority of processes, we have discussed in Lecture 1
  - When a process is killed, all the components running inside are killed

Threads

- As there is only one thread of execution, both the application components and UI interactions are done in sequential order
- So a long computation, I/O, background tasks cannot be run directly into the main thread without blocking the UI
- If your application is blocked for more than 5 seconds, the system will display an “Application Not Responding" dialog
  - leads to poor user experience

Threads - continued

- UI functions are not thread-safe in Android
- You can only manipulate the UI from the main thread
- So, you should:
  - Dispatch every long operation either to a service or a worker thread
  - Use messages between the main thread and the worker threads to interact with the UI

Working with Threads

- There are several ways of implementing worker threads in Android:
  - Use the standard Java threads, with a class extending `Runnable`
    - You need to do messaging between your worker thread and the main thread
    - Messages are possible through handlers or through the `View.post` function
  - Use Android's `AsyncTask`
    - `AsyncTask` has four callbacks: `doInBackground`, `onPostExecute`, `onPreExecute`, `onProgressUpdate`
    - Only `doInBackground` is called from a worker thread
    - Others are called by the UI thread
  - More sophisticated approaches are based on the `Loader` class, retained Fragments and services

Thread with Runnable - Example

- Observe the `View.post`

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

http://developer.android.com/guide/components/processes-and-threads.html

What we have learnt?

- What is Android
- Lifecycle management of Android applications
- How to develop GUI in Android
- Basic application components
  - Activities, Intents, BroadcastReceivers, Content Providers, Services, Threads
- So you are ready for developing Android applications !!!

Home Assignment - 1

- Contact picker
  - Have an activity with design in fig-A with contacts of the phone
  - Select a contact
  - Send an email to the selected contact
  - Back to original screen and display as in fig-B
  - Display the contact details of selected one
    - Name, Phone no, email
  - Have an action bar and introduce search functionality

A

B

Deadline 30th September 2015

Next week

- Mobile Application Development with iOS
- We get back to Android again a bit later

THANK YOU