Android Programming Topics

Oct. 29/Oct. 31, 2012

- **Android Storage Options**
- **Networking Connection Support Methods**
- **Download Manager Class**
- **Using Internet Services**
- **Connecting to Google App Engine**
- **Working with SQLite Databases**
- **Creating Content Providers**
- **Services:** Intent Monitoring services (using repeating Alarms to Schedule Network Refresh); Intent Services – Update Service
- **Alarms** (at predetermined times or intervals): Fire broadcast Intents, Start services, Open activities
- **Notifications** (to alert users of events that may require their attention without one of its Activity being visible)

Android Storage Options, http://developer.android.com/guide/topics/data/data-storage.html

- **Shared Preferences** class (store private primitive data int key-value pairs), http://developer.android.com/guide/topics/data/data-storage.html#pref
  - **getSharedPreferences()**
  - **getPreferences()**
  - To write values
    - Call **edit()** to get a **SharedPreference.Editor**
    - Add values with methods such as **putBoolean()** and **putString()**
    - Commit the new value with **commit()**
  - To read values
    - Use **getBoolean()** and **getString()**
- **Internal Storage** (store private data on the device memory)
  - To create and write a private file to the internal storage
    1) Call **openFileOutput()** with the name of the file and the operating mode. This returns a **FileOutputStream**
    2) Write to the file with **write()**
    3) Close the stream with **close()**
  - To read file from internal storage
    1) Called **openFileInput()** and pass it the name of the file to read. This returns a **FileInputStream**
    2) Read bytes from the file with **read()**
    3) Close the stream with **close()**
  - Saving Cache Files (cache some data)
    - Use **getCacheDir()** to open a File that represents the internal director where your application should save temporary cache files
  - Other related methods
    - getFilesDir()
    - getDir()

        - deleteFile()
        - fileList()
- **External Storage** (store public data on the shared external storage)
    - This can be a removable storage media (SD card) or an internal (non-removable) storage.
    - Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.
    - Checking File Availability
        - Should always call **getExternalStorageState()** to check whether the media is available. The media might be mounted to a computer, missing, read-only, or in some other state.
    - Accessing Files on External Storage
        - Use **getExternalFileDir()** to open a File (API Level 8 of greater)
            - DIRECTORY_MUSICS, DIRECTORY_PICTURES, DIRECTORY_RINGTONES, DIRECTORY_PODCASTS, DIRECTORY_NOTIFICATIONS, and DIRECTORY_ALARMS
    - Saving Files that Should be Shared
    - Saving Cache Files
- **SQLite Databases** (store structured data in a private database)
    - Full support for SQLite databases for creating, opening, and upgrading databases
    - Public abstract class **SQLiteOpenHelper** extends Object, http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html
        - A helper class to manage database creation and version management
            - Database names
            - Database tables
            - Database version
        - Write/read from the database: **getWritableDatabase(), getReadableDatabase()**
    - public final class **SQLiteDatabase** extends SQLiteClosable, http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html
        - for Opening and Creating Databases without the SQLite Open Helper
        - Use Context's openOrCreateDatabase method SQLiteDatabase db = Context.openCreateDatabase(DATABAE_NAME, Context.MODE_PRIVATE, null)
        - Public Methods
            - void beginTransaction()
            - void beginTransactionNonExclusive()
            - void beginTransactionWithListener (SQLiteTransactionListener transactionListener)
            - SQLiteStatement compileStatement(String sql)
            - static SQLIteDatabase create(SQLiteDatabase.CursorFactory factory) – create a memory backed SQLite database
            - int delete(String table, String whereClause, String[ ] whereArgs) -- deleting rows in the database
            - static boolean deleteDatabase(File file) – deleting a database including its journal files and other auxiliary files
            - …
            - void endTransaction()
            - vod execSQL(String sql) – execute a SQL statement that is NOT a SELECT/INSERT/UPDATE/DELETE
            - …

- Long getMaximumSize() – returns the current database page size, in bytes
- long getPageSize() -- gets current database page size, in bytes\final String getPath() – gets the path to the database file
- int getVersion() – get the database version
- Boolean intTransaction() -- returns true if the current thread has a transaction pending
- Long insert(String table, String nullColumnHack, ContentValues values) – inserting a row into database
- Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String orderBy)
- …

  - SQLiteQueryBuilder – for used in complex queries
  - Cursor – points to all the rows found by the query
- **Network Connection** (store data on the web with your own network layer)
  - When it's available
  - Store and retrieve data on Web-based services
  - import japa.net.*
  - import android.net.*

SQLite database and Content Providers

- SQLite Database, http://www.sqlite.org/

Native content provider examples

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

Content Providers

- Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.
- ContentResolver class, http://developer.android.com/reference/android/content/ContentResolver.html
  - Provides applications access to the content model

Android' Networking Connection Methods

- Two Connecting Methods
  - Mobile Internet Access
    - GPRS (General Packet Radio Service) on 2G and 3G cellular GSM network
    - EDGE (Enhanced Data rates for GSM Evolution)
    - 3G, 4G, LTE
  - Wi-Fi
- HTTP Internet Access - Connecting to an Internet Resources
  - public static final class Manifest.permission extends Object, http://developer.android.com/reference/android/Manifest.permission.html
  - Need to add an **INTERNET** uses-permission node to <manifest … /manifest>
    - <uses-permission android:name="android.permission.INTERNET"/>

- public abstract class **URLConnection** extend Object, http://developer.android.com/reference/java/net/URLConnection.html
- public abstract class **HttpURLConnection extends URLConnection**, http://developer.android.com/reference/java/net/HttpURLConnection.html

- Network Connectivity and Status - public class **ConnectivityManager** extends Object, http://developer.android.com/reference/android/net/ConnectivityManager.html
  - Query info about the state of network connectivity with the following responsibilities
    - Monitor network connection such as Wi-Fi, and cellular network: GPRS, UMTS – Universal Mobile Telecommunication System: 3G, etc
    - Send broadcast intents when network connectivity changes
    - Attempt to "fail over" to another network when connectivity to a network is lost
    - Provide APIs that allows application to retrieving network status
  - Types of Connectivity
    - TYPE_BLUETOOTH - default Bluetooth connection
    - TYPE_DUMMY - dummy data connection
    - TYPE_ETHERNET - default Ethernet connection
    - TYPE_MOBILE - default mobile connection
    - TYPE_MOBILE_DUN - same as TYPE_MOBILE; used by applications performing Dial-Up Networking so that the carrier is aware of DUN traffic.
    - TYPE_MOBILE_HIPRI - same as TYPE_MOBILE, high priority data connection
    - TYPE_MOBILE_MMS - Multimedia Messaging Services
    - TYPE_MOBILE_SUPL - for talking to carrier's Secured User Plane Location for locating the device
    - TYPE_WIFI
    - TYPE_WIMAX
  - Methods
    - NetworkInfo **getActiveNetworkInfo**()
    - NetworkInfo[ ] getAllNetworkInfo()
    - NetworkInfo getNetworkInfo(int networkType)
    - Int getNetworkPreference()
    - boolean isActiveNetworkMetered()
    - static Boolean isNetworkTypeValid(int networkType)
    - boolean requestRouteToHost(int networkType, int hostAddress)
    - void setNetworkPreference(int preference)
    - int startUsingNetworkFeature(int networkType, String feature)
- public class **NetworkInfo** extends Object implements Parcelable, http://developer.android.com/reference/android/net/NetworkInfo.html
- Example [3]

```
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
…
//…
connectivityManager cm = (connectivityManager) getSystemService(Context.CNNECTIVITY_SERVICE);
NetworkInfo ni = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWiFiAvail = ni.isAvailable();
boolean isMobileConn = ni.isConnected();
status.setText("WiFi\n Available = "+ isWifiAvail + "\nCoo = " + isWificonn + "\n\Mobile\n Avail = " +
isMobileAvail + "\nConn = " + isMobileConn);
```

Download Manager

- public class **DownloadManager** extends Object, http://developer.android.com/reference/android/app/DownloadManager.html
- A system Service that handles long-running the HTTP download.
- Clients may request that a URI be downloaded to a particular destination file. The download manager will conduct download in the background, taking care of HTTP interactions and retrying downloads after failures or across connectivity changes and system reboot.
- Methods
    - getSystemService()
- Downloading Files
- Download Manager Notifications
- Download Location
- Cancelling and Removing Downloads
- Querying the Download Manager

An Example:

```
String serviceString = Context.DOWNLOAD_SERVICE;
DownloadManager myDownloadManager;
myDownloadManager = (DownloadManager) getSystemService(serviceString);
Uri myUri = Uri.parse(http:////developer.android.com/sharable/icon_templates-v4.0.zip);
DownloadManager.Request request = new Request (myUri);
Long reference = myDownloadManager.enqueue(request);
….
```

Using Internet Services

- Software-as-a-Service (SaaS) – cloud computing services
- Google API Services, https://developers.google.com/apps-script/googleapisservices
- Google Geocoding API, https://developers.google.com/maps/documentation/geocoding
- Yahoo! Pipes, http://pipes.yahoo.com/pipes/
- Google App Engine, https://developers.google.com/appengine/
    - Store and retrieve data related to a particular user
    - Use Account Manager to handle the authentication
- Google Cloud Platform, https://cloud.google.com/products/
- Amazon Web Service, http://aws.amazon.com/

Connecting to Google App Engine

- Google Plugin for Eclipse: App Engine Connected Android Support, https://developers.google.com/eclipse/docs/appengine_connected_android
- Need to add an uses-permission node to <manifest … /manifest>
    - <uses-permission android:name=”android.permission.**GET_ACCOUNTS**”/>
- public class **AccountManager** extends Object, http://developer.android.com/reference/android/accounts/AccountManager.html
- Three steps
    - Request an authentication token

- Use the authentication token to request a authentication code
- Use the authentication cookie to make authenticated requests

Alarms

- public class AlarmManager extends Object, http://developer.android.com/reference/android/app/AlarmManager.html
- Provide access to the system alarm services which can be used to schedule your application to run at a specific time
- Public Methods
  - Creating, Setting, and Cancelling Alarms
    - void cancel(PendingIntent operation) – remove an alarms with a matching Intent
    - void set(int type, long triggerAtMillis, PendingIntent operation) – schedule an alaram
  - Setting Repeating Alarms
    - void setInexactRepeating(int type, long triggerAtMills, long intervalMillis, PendingIntent operation)
    - Constants
      - INTERVAL_FIFTEEN_MINUTES
      - INTERVAL_HALF_HOUR
      - INTERVAL_HALF_DAY
      - INTERVAL_DAY
  - Using Repeating Alarms to Schedule Network Refreshes
    - void setRepeating(int type, long triggerAtMills, long intervalMills, PendingIntent operation)

Notifications

- To alert users of events that may require their attention in the following forms
  - Status bar: a persistent icon goes in the status bar
  - Light: Flashing LEDs on the device
  - Playing audible sounds or vibrating (ringtones, Media Store audio)
  - Details display within the extended notification tray
- public class **NotificationManager** extends Object, http://developer.android.com/reference/android/app/NotificationManager.html
  String svcName = Context.NOTIFICATION_SERVICE;
  NotificationManager myNotificationManager;
  myNotificationManager = (NotificatiobManager) getSystemService(svcName);
  …
  Int icon = R.drawable.icon;
  String tickerText = “Important Notification: Please Read your Email”;
  long when = System.currentTimeMillis();
  …
  Notification myNotification = new Notification(icon, tickerText, when);
  …
- public class Notification extends Object implements Parcelable, http://developer.android.com/reference/android/app/Notification.html
    - Notification.DEFAULT_ALL
    - Notification.DEFAULT_LIGHTS
    - Notification.DEFAULT_SOUND

- Notification.DEFAULT_VIBRATE

References

[1] Android Developer Reference: API Classes, http://developer.android.com/reference/packages.html

[2] **Professional Android 4 Application Development**, 2012, by Reto Meier, published by John Wiley & Sons, Inc., Indianapolis, Indiana.

[3] Android Wireless Application Development, Volume II: Advanced Topics, by Lauren Darcy and Shane Coder, 3rd Edition, 2012, published by Addison-Wesley