DO WHAT THE FUCK YOU WANT

Telephony API Packages (12.10)

```
android.telephony.TelephonyManager
```

SQLite Command to View All Tables

inside the SQLite Command Line type :

```
.tables
```

SQLiteQueryBuilder with it's Public Methods

SQLiteQueryBuilder to build query without knowing the query. SQLiteQueryBuilder.setTables(string tables) to declare the tables, multiple can be mentioned. to create table, use only raw sql (CREATE TABLE bleh bleh)

Content Providers

Use Custom content providers to share data between different apps, as you can't do that without this. the code is fucking long i dont want to write it, ty. (Page 6.22)

Shared Preferences

Store a key value data into an xml files usually stored in

```
/data/data/YOUR_PACKAGE_NAME/shared_prefs/YOUR_PREFS_NAME.xml
```

the code to edit and create (if not exists) shared preferences is

```
String myPrefs = "myPrefs";
SharedPreferences sp = getSharedPreferences(myPrefs, Activity.MODE_PRIVATE);
ShatedPreferences.editor edit = sp.edit();

edit.putInt("key", value);
edit.commit();
```

to retrieve is declare the above sharedpref object and then write :

```
String dataFromSharedPrefs = sp.getInt("key",specifyDefaultValue);
```

Testing Consideration

If you are not stupid enough to develop an ugly ass android apps, takes this into consideration:

1. **Change in Orientation**, users can flip the phone into whatever the fucks they want, so please ensure your apps :
    - The UI is redrawn quickly when user changes orientation

    - The apps maintain its state during orientation changes and does not lose user input that have been entered
2. **Change in the device's configuration**, check whether the apps can adapt, improvise, overcome the changes within the device configuration.
3. **Battery Life**, Make sure your apps only uses minimal amounts of battery and Runs smoothly inside **KAPTEN's** phones for at least 2000 Hours
4. **Dependence on external resources**, If your apps uses external resources such as internet connection, gps, etc. you should test your apps behavior and performance in the absence of these resources (e.g notifying the user that their internet is disconnected), so your apps can be used for poor fucks man.

EULA

Eula is End User License Agreement, it must be added before you publish your apps in store. Steps Before :

- Testing the app on actual devices

After:

- Add Licensing Support

Further Reading : Page 15.9

Android Manifest (Read Code Comment)

not a virus

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Package name-->
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.myapp">
    <!--Permission-->
    <uses-permission android:name="android.permission.SEND_SMS"/>

    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- Activity -->
        <activity android:name=".MainActivity">
            <intent-filter>
                <!--Intent Filter-->
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

Codes inside Application Tag :

- `<activity>` for each subclass of Activity.
- `<service>` for each subclass of Service.
- `<receiver>` for each subclass of BroadcastReceiver.
- `<provider>` for each subclass of ContentProvider.

Activity Lifecycle

- **onCreate** called when activity is first created.
- **onStart** called when activity is becoming visible to the user.
- **onResume** called when activity will start interacting with the user.
- **onPause** called when activity is not visible to the user.
- **onStop** called when activity is no longer visible to the user.
- **onRestart** called after your activity is stopped, prior to start.

- **onDestroy** called before the activity is destroyed.

Intent

- **Implicit Intents** start a component from another apps or from system (like dialing a call, opening xnxx.com in the browser, etc.)
- **Explicit Intents** start a component from our apps, typically starting another activity.

Intent Example :

Implicit

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.hentaimama.com"));
startActivity(intent);
```

Explicit

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

Activity Phase

See Activity Lifecycle, pretty much the fucking same.

DDMS Perspective

DDMS stands for Dalvik Debug Monitor Server. It gives the wide array of debugging features:

1. Port forwarding services
2. Screen capture
3. Thread and heap information
4. Network traffic tracking
5. Location data spoofing

Static Files As resources using `Android.content.res.Resources`

Used when you want to include large or pre existing files as a resouce for your app instead of a database.

usage :

```
Resources myRes = getResources();
InputStream files = myRes.openRawResource(R.raw.myfilename);
```

ADB

Android Debug Bridge, allow to interacts with device/emulator using command line

Command : `adb [-d|-e|-s <phone serial>] <command>`

example : `adb devices` , `adb start-server` , `adb kill-server`

Progress Dialog

We can display the android progress bar dialog box to display the status of work being done e.g. downloading porn, analyzing status of work etc.

usage:

```
ProgressDialog progressBar = new ProgressDialog(this);
progressBar.setCancelable(true);//you can cancel it by pressing back button
progressBar.setMessage("Porn downloading (°ʖ̯°)....");
progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressBar.setProgress(0);//initially progress is 0
progressBar.setMax(100);//sets the maximum value 100
progressBar.show();//displays the progress bar
```

SMS and Call with Intent ™

Snip Read it yourself at Page 12.3

Android Building Block

Too General, Read page 2.15

Ordered Broadcasts

Ordered Broadcast is the type of broadcast which is sent in a synchronous manner i.e. one by one to each listener. (Broadcast Definition see Page 8.15)

example:

```
private static String Log_Tag = "SendOrderedBroadcastActivity";
private static String Action = "com.example.mypackage";
private static String Extras = "Breadcrumb";

Intent intent = new Intent(Action);
        sendOrderedBroadcast(intent, null, new BroadcastReceiver() {
            @SuppressLint("NewApi")
            @Override
            public void onReceive(Context context, Intent intent) {
                Bundle results = getResultExtras(true);
                Log.i(Log_Tag, "Final Result Receiver = " + results.getString(Extras,
"nil"));
            }
        }, null, Activity.RESULT_OK, null, null);
```

Bitmap, Paint, Canvas, View

- Canvas (Package `android.graphics.Canvas`) represents a surface where you can draw shapes and images
- Bitmap (Package `android.graphics.Bitmap`) When you create `Canvas` object it is necessary to create atleast one `Bitmap` Object, which holds the pixel data for the graphics.
- Paint (Package `android.graphics.Paint`) provides information about the style & colors for drawing a symbols, text, and graphics

- view (Package `android.view.View`) simple graphics and animation that does not requires high framerate. Must extend this class to draw crap ex: `class Panel extends View`

Layout Inflater

Get an Element from Another Layout XML.

When you use a custom view in a `ListView` you must define the row layout. You create an xml where you place android widgets and then in the adapter's code you have to do something like this:

```
public MyAdapter(Context context, List<MyObject> objects) extends ArrayAdapter {
  super(context, 1, objects);
  /* We get the inflator in the constructor */
  mInf = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
  View view;
  /* We inflate the xml which gives us a view */
  view = mInflater.inflate(R.layout.my_list_custom_row, parent, false);

  /* Get the item in the adapter */
  MyObject myObject = getItem(position);

  /* Get the widget with id name which is defined in the xml of the row */
  TextView name = (TextView) view.findViewById(R.id.name);

  /* Populate the row's xml with info from the item */
  name.setText(myObject.getName());

  /* Return the generated view */
  return view;
}
```

`LocationManager.NETWORK_PROVIDERS`

Represents the name of the network location provider, point the location of cell towers and/or wi-fi. Results are retrieved by means of network lookup. The providers Requires the `android.permission.ACCESS_COARSE_LOCATION` or `android.permission.ACCESS_FINE.LOCATION`.

Other Options are `LocationManager.GPS_PROVIDERS`, it uses GPS for providing location data.

TOAST

to notify user, takes only small amount of screen, and will fade eventually (Package `android.widget.Toast`)

example:

```
Toast toast = Toast.makeText(getApplicationContext(),
        "This is a message displayed in a Toast",
        Toast.LENGTH_SHORT);

toast.show();
```

not a virus

Making an instance of TelephonyManager

```
public abstract Object getSystemService(String name)
```

Content Provider in AndroidManifest.XML

Wrap it inside `<application>` tag

```
<provider
   android:name=".provider.LentItemsProvider"
   android:authorities="de.openminds.samples.cpsample.lentitems"
   android:exported="true"
   android:grantUriPermissions="true"
   android:label="LentItemsProvider"
   android:readPermission="de.openminds.samples.cpsample.lentitems.READ"
   android:writePermission="de.openminds.samples.cpsample.lentitems.WRITE" />
```

Custom Provider

After creating the custom provider, register it in AndroidManifest.xml.

```
<provider
   android:name="myCustomProvider"
   android:authorities="com.DatabaseDemo.provider.myCustomProvider">
</provider>
```

View Listener

- `onClick()` From `View.OnClickListener`. This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.
- `onLongClick()` From `View.OnLongClickListener`. This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).
- `onFocusChange()` From `View.OnFocusChangeListener`. This is called when the user navigates onto or away from the item, using the navigation-keys or trackball.
- `onKey()` From `View.OnKeyListener`. This is called when the user is focused on the item and presses or releases a hardware key on the device.
- `onTouch()` From `View.OnTouchListener`. This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).
- `onCreateContextMenu()` From `View.OnCreateContextMenuListener`. This is called when a Context Menu is being built (as the result of a sustained "long click").

EXIT SHELL LINUX

exit/stop command use `ctrl+d` exit shell use command `exit`

or find the pid and then kill it `ps -ax | grep shell_name` and `pkill -9 PID_of_shell`

Testing Tools in Android

ADT - Android Debugging Tools Consists of these tools :

- ADB - Android Debug Bridge, Versatile command line tools allows to communicate with an android-enabled devices/emulator
- DDMS - Dalvik Debug Monitor Server, GUI Tools allows to communicate with android
- Device / AVD, Android Virtual Devices, Simulate how your apps will be running and behave on a real device.

Media Players

Method	Description
public void setDataSource(String path)	sets the data source (file path or http url) to use.
public void prepare()	prepares the player for playback synchronously.
public void start()	it starts or resumes the playback.
public void stop()	it stops the playback.
public void pause()	it pauses the playback.
public boolean isPlaying()	checks if media player is playing.
public void seekTo(int millis)	seeks to specified time in miliseconds.
public void setLooping(boolean looping)	sets the player for looping or non-looping.
public boolean isLooping()	checks if the player is looping or non-looping.
public void selectTrack(int index)	it selects a track for the specified index.
public int getCurrentPosition()	returns the current playback position.
public int getDuration()	returns duration of the file.
public void setVolume(float leftVolume,float rightVolume)	sets the volume on this player.

Location Based Service

LBS provide information about your device's current location. LBS Get location data from these location providers.

- GPS
- Cell Tower Triangulation
- Public Wi-Fi Hotspots

Logs

- `Log.e(String, String)` (error)
- `Log.w(String, String)` (warning)
- `Log.i(String, String)` (information)
- `Log.d(String, String)` (debug)
- `Log.v(String, String)` (verbose)

Resource Manager, Notification Manager, Activity Manager

- Resource Manager : Manage Localized Resources (e.g Strings, Graphics, Layout, etc.)
- Notif Manager ; Enable Apps to Display Messages in the Notif Bar
- Activity Manager : Manages activities Local Files

Building Block of an Android Application

- **Activity** is a **USER INTERFACE** of an Android Apps
- **Services** is an app that runs in the background
- **Content Providers** allow apps to retrieve, modify or share data from another source, but require permission from the content provider.
- **Broadcast Receivers** is a component that respond to a system event, such as the screen being turned off, low battery, etc.

Activity States

- **Runnning** The activity is visible to the user on the screen and user can interact with the activity
- **Paused** Another activity is visible to the user.
- **Stopped** The activity is completely obscured by another activity

Dialog Subclass

- `AlertDialog` create an alert dialog, can contain up to three buttons, and/or a list of selectable items with radio/checkbox
- `ProgressDialog` create a progress loading dialog, extended from Alert dialog, contains same feature as it.
- `DatePickerDialog` allow user to pick a date
- `TimePickerDialog` allow user to pick a time

`onDestroy()` and `stopSelf()`

Call `stopSelf()` to stop a Service. After you call it, the Android Framework will call `onDestroy()` method automatically.

Proprietary means Closed Source

- Android is non-proprietary (Uses GNU-GPL v2 License)
- Symbian, iOs, and BBOS is a Proprietary Software

Test-case

You need to extend the testing junit class with involved Activities in order to create test case class, also import android.test

NameClass: MainActivityTest

Involved : MainActivity

Using TestClass : ITestYourAss

not a virus

```
package com.aweawe.testing;
import com.awewe;
import android.test.ITestYourAss;
public class MainActivityTest extends ITestYourAss<MainActivity> {
}
```

SQLiteQueryBuilder create table and columns

note : SQLLiteQueryBuilder is used to GET data, And show it to E.g : tables, listview, etc. not to Create Tables

usage :

```
    SQLiteQueryBuilder builder = new SQLiteQueryBuilder(); //instance creation
    builder.setTables("YourTableThatWantsToBeShowed");
    //equivalent as : SELECT * FROM YourTableThatWantsToBeShowed

   String columns[] = new String[] {"column1","column2"};
   //equivalent as : SELECT column1, column2 FROM YourTableThatWantsToBeShowed
    cursor c = sqlBuilder.query(DBCon, columns, null, null, null, null);
```

show all query

```
public Cursor queryAll(Context context, Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
    queryBuilder.setTables(getTableName());

    Cursor cursor = queryBuilder.query(mDB, projection, selection, selectionArgs, null,
null, sortOrder);
    cursor.setNotificationUri(context.getContentResolver(), uri);
    return cursor;
}
```

Pages that is not covered on the rangkuman

- Content Provider - Pg. 6.22
- Intent for Dialing and Sending SMS - Pg. 12.3 - 12.10
- SQLiteQueryBuilder for Creating Table and Column - Unknown Pages
- Telephony Calls - Pg. 12.10

This rangkuman is only intended for learning, the author didnt authorize nor responsible for those who caught cheating while in exam. Also the author did not responsible for the sins™ that you have make for cheating in exam. the author hopes that this rangkuman will help you learn the materi for the exam, thank you.