**Android Versions** (5, 85)

**AIDL** (152)

**array**(24)

**attributes**(27, 53

**adb** (40, 47, 99)

**aapt** (48)

**Activity** (84)

**Activity Methods** (62)

**Atomic** (127)

**Bundle** (94)

**button program** (32)

**color**(22)

**context** (43, 47)

**dimension resource** (29)

**debug** (38)

**DDMS** (42)

**EditText**(52, 57)

**event handling** (58)

**Google Manual (signing)** (159)

**id**(25)\*

**intent**(61, 76-83, 92)

**inner anonymous class** (37)

**Keytool** (161)

**Log** (38, 61)

**Life Cycle** (71)

**Main.xml**(65)

**Main Application Thread** (127)

**Messages** (134)

**Music/Sound app** (64)

**Multi-Threading** (125, 138)

**RadioButtons** (49, 60)

**SD Card** (97-102)

**Sensors**(166)

**Services** (149)

**Setup** (1

**setContentView**(13)

**storage devices** (103)

**string**(20)

**sub-directories** (9-11) \*

**SQLite(**113)

**Tags/attributes**(15-17)

**TextView**(25)

**Toast** (43, 45)

**Zipalign**(165)

cursor c = rawQuery(select \* from xxx”, null);   
//use environment getData directory to check for db

    boolean isTableEmpty()   
    {   
           Cursor c = this.getReadableDatabase().rawQuery(   
                "Select count(\*) from xxx", null);   
        while(!c.moveToNext())   
        {   
            int numRows = c.getInt(0);   
            if (numRows<1)   
            {   
                return false;   
            }   
            return true;   
        }   
           return false;   
    }

boolean isDBExists()

{

//get env variable

String path = “DB located DATA/data/APP\_NAME/databases/Dbname”;

File f = new File(path);

if(f.exists()) return true;

return false;

}

SQLiteOpenHelper \*\*\*  
import android.content.ContentValues;  
import android.content.Context;  
import android.database.sqlite.\*;  
  
  
public class DbDemoActivity extends SQLiteOpenHelper {  
    /\*\* Called when the activity is first created. \*/  
    private static final String DBNAME="mydb";  
    private static final String NAME="name",AGE="age";  
      
    public DbDemoActivity(Context c){  
        super(c,DBNAME, null,1);  
    }  
      
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL("create table info(-id integer primary key" +  
                "autoincrement,name text,age integer);");  
        ContentValues cv = new ContentValues();  
        cv.put(NAME, "dave");  
        cv.put(AGE, 19);  
        db.insert("info", null, cv);  
        cv.put(NAME, "julie");  
        cv.put(AGE, 18);  
        db.insert("info", null, cv);  
   
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // TODO Auto-generated method stub  
          
    }  
}

0. Data Storage

1. Shared preferences

     SharedPreferences sp = getSharedPreferences("myfile.txt", 0)

                      // 0 mode: limited access

     String myname = sp.getString("name", "none");

                // name:key, none: default v

                // there are many get... methods available

     Float f = sp.getFloat("weight", 0.0);

                //  getBoolean, getLong, getInt... and so on...

     ex) if((myname=sp.getString("name","xxx").equals("xxx")))... ???????

     SharedPreferences.Editor ed = sp.edit();

               // Editor in charge of writing files.

     ed.putString("myCar", "Honda");

     // apply(v1.5) or commit(v2) == flush

     ed.commit(); // OR ed.apply();

     ed.clear(); // clear the file

     ed.putBoolean(k,v);

     ed.putInt(k,v);

     ed.putString(k,v);

     ed.remove(k);

     if(ed.contain(k))...

     Map<String,?> getAll();   //retrieves all the value (generic)

2. Internal Storage

 String str="This line sill be saved in the file.";

     FileOutputStream fos = openFileOutput("myfile",

                         Context.MODE\_PRIVATE);// destructive filemode

                         Context.MODE\_APPEND); // appending

                  Context.MODE\_APPEND|Context.MODE\_WORLD\_READABLE); //

                  Context.MODE\_APPEND|Context.MODE\_WORLD\_WRITABLE);

     fos.write(str.getBytes()); //destructive

     fos.close();

     FileInputStream fis = openFileInput("myfile");

 //     int fis.read(byte buf[], int offset, int byteCount);

 //    returns number of lines fis read

 //     if it returns -1 : EOF

 // also can use Scanner

3. External Storage

 // default : world readable

   // first thing to do : is the device available?

   // 1)

   boolean storageAvailable = false;

   boolean storageWritable = false;

   String states = Environment.getExternalStorageStates();

   if(Environment.MEDIA\_MOUNTED.equals(state))

       storageAvailable=storageWritable=true;

   else if(Environment.MEDIA\_MOUNTED\_READ\_ONLY.equals(state))

       storageAvailable=true;

   // 2)

   // bool isExternalStorageremovable()

   // 3)

   void createExternalStoragePrivateFile()

   {          // null:file will be created in the root of the device

        File f = new File(getExternalFilesDir(null),"Demo.jpg");

        try

        {

            InputStream is = getResources().openRawResource(R.drawable.baloons);

            Outputsteam os = new FileOutputStream(f);

            byte data[] = new byte[is.available()];

            is.read(data);

            os.write(data);

            is.close();

            os.close();

         }catch(IOException e){Log.w("ExternalStorage", "Err writing"+f,e)}

    }

    void deleteExternalStoragePrivateFile()

    {

        File f = new File(getExternalFilesDir(null),"Demo.jpg");

        if(f!=null) f.delete();

    }

    boolean hasExternalStoragePrivateFile()

    {

        File f = new File(getExternalFilesDir(null),"Demo.jpg");

        if(f!=null) return(f.exists());

        return(false);

    }