**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)

**Questions:**

**What is Activity? (12) \* p31**

-a UI concept that usually represents a single screen in your application. Something that helps the user do one thing, which can be viewing data, creating data, or editing data.

**What is a View?** **(p31)**

-Views are UI elements that form the basic building blocks of a user interface. View can be a button, a label, a text field, etc.

**What is Intent?** **(p32)**

-Best to describe this is by thinking that intent generically defines an “intention” to do some work. For example, broadcast a message, start a service, launch an activity, display webpage, dial a phone number, etc.

-also used by application to notify of specific events

-not always initiated by your application

-can be explicit or implicit

**AndroidManifest.xml** **(p33) 11**

-defines contents and behavior or your application. (activities, services, permissions)

How to remove Activity name? (95, 63)

**What is ash? (1)**

-android shell, command line terminal

**Why not use .class? (1)**

-class files are not economic or efficient (memory consumption); we use .dex file (more optimized)

**What is skin? (3)**

-resolution

**What must be in system path?**

/tools and /platform-tools

**Why are all resources written in xml? (6)**

it is simple to use/follow; if we change data we do not need to re-compile; if we sell the app then user can modify resource file

**What is inside bin? (14)**

-two sub directories: classes (all classes created in android program .class files – R.class, JavaApp.class) and classes.dex (all dalvik files are in here, JavaApp.apk\_ (zip file of all gui resources), JavaApp-debug-unalinged.apk, JavaApp-dubug.apk (condensed app ready to be sold for use) ).

**Where to find main.xml? (17)**

res/layout/main.xml

**How to setup a project? (7)**

**What is R.java? (4, 18)**

-repository, has a reference to all of our resources; auto-generated; first line in R.java is package name

**Where is R.java? (10, 13, 18)**

-gen/my/package/R.java (my/package is package name)

-bin (in backup)

**How to put strings into resource files? (19)**

res/values/strings.xml

**Two ways to get value of a String? (20)**

1. in R.java , in res/values/strings.xml

**Two ways to see variables?(19)**

1. source code ?? (I think in xml)

2. R.java

**How to add colors to an application? (22)**

1. res/values/color.xml

2. in java code (but should not do this)

**Difference between TextView and Buttons? (28)**

TV – no activity behind it

Button – something happens when a button is selected; an object is created; we must catch this object

**Why System.out.print() not displayed on screen? (39)**

it is text (character type) and we need graphics type to display on screen; use log to see text

**Standalone program?**

in AndroidManifest.xml under application tag add android:debugable =”true”?

allows you to debug application on cellphone

**Three ways to display message?**

Toast, log, Dialogs (error dialog, alert dialog)

**What is memory leak? (44)**

-when we create a lot of resources and never remove them (run out of memory)

**Better debug tool? (63)**

-get from MOTODEV STUDIO; import android.os.Debug;

-Debug.startMethodTracing(“trace-filename”);

-Debug.stopMethodTracing();

**What is Context?** (don’t have a lot of info on this) (47)

-interface allowing access to applications specific resources

-a reference to the current object

-allows access to information about application environment

**How to programmatically know if emulator is running? (68)**

if(System.Environment.DeviceType != DeviceType.Emulator)

{

Log.e(Tag, “Emulator is not running”);

}

**How to check if application will run in cellphone? (68)**

if(“google\_sdk”.equals(Build.PRODUCT) )

{

on phone // if false the on emulator

}

google\_sdk or skd

what if not both?

String android\_id = Secure.getString(getContentResolver(), Secure.ANDROID\_ID);

if(android\_id == null)

{

we are in emulator

}

else

{

we are in phone/device

}

**What device are you running? (70)**

public boolean isEmpulator()

{

return Build.MANUFACTURER.equal(“unknown”);

}

**How do we know if an Activity calls our class?** (84)

1. In ManifestFile .LAUNCHER or .ALTERNATIVE

2. this.getIntent().getData(); //retyrns Uri of your file (name of program that called it)

if(this.getIntent().getData() == null) //if true then you are the launcher

**Two ways to know if the activity is the launcher** (84)

1. android manifest

2. java code

**How to get Android Version (85)**

-Android.os.Build.VERSION.SDK (1.5 or less)

-Android.os.Build.VERSION.SDK\_INT (above 1.5)

-Android.os.Build.VERSION\_CODES.FROYO

**How our program knows which version I am using?** (85)

-String versionName = getPackageManager().getPackageInfo(getPackageName(), 0).versionName;

-int versionCode = ‘ ‘ .versionCode;

NOTE:

in ManifestFile you can see version code and version number

**How to hid Activity Title in Application? (63)**

-inside onCreate() method add the following requestWindowFeature(Window.FEATURE\_NO\_TITLE);

-must also import android.view.Window;

Get info

-annotations (12)

C:\Users\Victor\Desktop\Word Documents\android\hw\_four\src\com\cs211d\hw

Android Debug Bridge version 1.0.26  
  
 -d                            - directs command to the only connected USB device  
                                 returns an error if more than one USB device is present.  
 -e                            - directs command to the only running emulator.  
                                 returns an error if more than one emulator is running.  
 -s <serial number>            - directs command to the USB device or emulator with  
                                 the given serial number. Overrides ANDROID\_SERIAL  
                                 environment variable.  
 -p <product name or path>     - simple product name like 'sooner', or  
                                 a relative/absolute path to a product  
                                 out directory like 'out/target/product/sooner'.  
                                 If -p is not specified, the ANDROID\_PRODUCT\_OUT  
                                 environment variable is used, which must  
                                 be an absolute path.  
 devices                       - list all connected devices  
 connect <host>[:<port>]       - connect to a device via TCP/IP  
                                 Port 5555 is used by default if no port number is specified.  
 disconnect [<host>[:<port>]]  - disconnect from a TCP/IP device.  
                                 Port 5555 is used by default if no port number is specified.  
                                 Using this ocmmand with no additional arguments  
                                 will disconnect from all connected TCP/IP devices.  
  
device commands:  
  adb push <local> <remote>    - copy file/dir to device  
  adb pull <remote> [<local>]  - copy file/dir from device  
  adb sync [ <directory> ]     - copy host->device only if changed  
                                 (-l means list but don't copy)  
                                 (see 'adb help all')  
  adb shell                    - run remote shell interactively  
  adb shell <command>          - run remote shell command  
  adb emu <command>            - run emulator console command  
  adb logcat [ <filter-spec> ] - View device log  
  adb forward <local> <remote> - forward socket connections  
                                 forward specs are one of:   
                                   tcp:<port>  
                                   localabstract:<unix domain socket name>  
                                   localreserved:<unix domain socket name>  
                                   localfilesystem:<unix domain socket name>  
                                   dev:<character device name>  
                                   jdwp:<process pid> (remote only)  
  adb jdwp                     - list PIDs of processes hosting a JDWP transport  
  adb install [-l] [-r] [-s] <file> - push this package file to the device and install it  
                                 ('-l' means forward-lock the app)  
                                 ('-r' means reinstall the app, keeping its data)  
                                 ('-s' means install on SD card instead of internal storage)  
  adb uninstall [-k] <package> - remove this app package from the device  
                                 ('-k' means keep the data and cache directories)  
  adb bugreport                - return all information from the device  
                                 that should be included in a bug report.  
  
  adb help                     - show this help message  
  adb version                  - show version num  
  
DATAOPTS:  
 (no option)                   - don't touch the data partition  
  -w                           - wipe the data partition  
  -d                           - flash the data partition  
  
scripting:  
  adb wait-for-device          - block until device is online  
  adb start-server             - ensure that there is a server running  
  adb kill-server              - kill the server if it is running  
  adb get-state                - prints: offline | bootloader | device  
  adb get-serialno             - prints: <serial-number>  
  adb status-window            - continuously print device status for a specified device  
  adb remount                  - remounts the /system partition on the device read-write  
  adb reboot [bootloader|recovery] - reboots the device, optionally into the bootloader or recovery program  
  adb reboot-bootloader        - reboots the device into the bootloader  
  adb root                     - restarts the adbd daemon with root permissions  
  adb usb                      - restarts the adbd daemon listening on USB  
  adb tcpip <port>             - restarts the adbd daemon listening on TCP on the specified port  
networking:  
  adb ppp <tty> [parameters]   - Run PPP over USB.  
 Note: you should not automatically start a PPP connection.  
 <tty> refers to the tty for PPP stream. Eg. dev:/dev/omap\_csmi\_tty1  
 [parameters] - Eg. defaultroute debug dump local notty usepeerdns  
  
adb sync notes: adb sync [ <directory> ]  
  <localdir> can be interpreted in several ways:  
  
  - If <directory> is not specified, both /system and /data partitions will be updated.  
  
  - If it is "system" or "data", only the corresponding partition  
    is updated.  
  
environmental variables:  
  ADB\_TRACE                    - Print debug information. A comma separated list of the following values  
                                 1 or all, adb, sockets, packets, rwx, usb, sync, sysdeps, transport, jdwp  
  ANDROID\_SERIAL               - The serial number to connect to. -s takes priority over this if given.  
  ANDROID\_LOG\_TAGS             - When used with the logcat option, only these debug tags are printed.

[[redirecting the log]]

ex)  
    std.out:   1>/dev/null  2>&1    # 1:system.out   2: system,err  
  
\*\* same way we can redirect this result to the Logcat.  
\*\* android.util.log  
  
\*\* Log.e()   : error log -takes care of err log (High priority)  
   Log.w()   : warning msg  
   Log.i()   : info msg  
   Log.d()   : debug msg  
   Log.v()   : verbose msg  (Low priority)  
  
\*\* Log.wtf()   : what a terrible failure.  (in above 2.2) Top of the list  
    => serious device malfunction  
    => sometimes.. emulator cause memory leak.  
  
=> these methods are overloadable

[[ Log.d ]]

    d(String tag, String msg) // overriding  
    d(String tag, String msg, throwable tr)  // overriding  
  
---------     // to narrow down the log  
    public static final String TAG = "Music Lovers";  
    Log.d(TAG,"i="+i);   // similar as 'grep'  
  
    try{  
  
    } catch(IOException e){  
         Log.d(TAG,"something wrong", e);    
    }  
  
[[ Log.e ]]  
    e(String tag, String msg)   
    e(String tag, String msg, throwable tr)  // overriding  
  
[[ Log.i ]]  
    i(String tag, String msg)   
    i(String tag, String msg, throwable tr)  // overriding  
  
[[ Log.v ]]  
    v(String tag, String msg)   
    v(String tag, String msg, throwable tr)  // overriding  
  
[[ Log.w ]]  
    w(String tag, String msg)   
    w(String tag, throwable tr) // overriding  
    w(String tag, String msg, throwable tr)  // overriding  
  
[[ Log.wtf ]]  
    v(String tag, String msg)   
    v(String tag, throwable tr) // overriding  
    v(String tag, String msg, throwable tr)  // overriding  
  
: WE WILL USE THEM VERY OFTEN  
: v never be compiled nor go to the final product.  
: d method compiled but not go to the final product.  
: rest of them will be compiled and go to the final product.

\*\*Delete Logcat

adb logcat -f fname  
adb logcat -c  
adb logcat filter(V,D,I,W,E,F:fatal=wtf)  
  
ex) adb logcat MyActivity:I MyApp:D  \*:S    
   # show info for MyActivity, show debug msg for MyApp and shut up for everthing else.  
  
    adb logcat -s  
   # always silent unless otherwise ....

======  ADB =====================

Android Debug Bridge version 1.0.26  
  
 -d                            - directs command to the only connected USB device  
                                 returns an error if more than one USB device is present.  
 -e                            - directs command to the only running emulator.  
                                 returns an error if more than one emulator is running.  
 -s <serial number>            - directs command to the USB device or emulator with  
                                 the given serial number. Overrides ANDROID\_SERIAL  
                                 environment variable.  
 -p <product name or path>     - simple product name like 'sooner', or  
                                 a relative/absolute path to a product  
                                 out directory like 'out/target/product/sooner'.  
                                 If -p is not specified, the ANDROID\_PRODUCT\_OUT  
                                 environment variable is used, which must  
                                 be an absolute path.  
 devices                       - list all connected devices  
 connect <host>[:<port>]       - connect to a device via TCP/IP  
                                 Port 5555 is used by default if no port number is specified.  
 disconnect [<host>[:<port>]]  - disconnect from a TCP/IP device.  
                                 Port 5555 is used by default if no port number is specified.  
                                 Using this ocmmand with no additional arguments  
                                 will disconnect from all connected TCP/IP devices.  
  
device commands:  
  adb push <local> <remote>    - copy file/dir to device  
  adb pull <remote> [<local>]  - copy file/dir from device  
  adb sync [ <directory> ]     - copy host->device only if changed  
                                 (-l means list but don't copy)  
                                 (see 'adb help all')  
  adb shell                    - run remote shell interactively  
  adb shell <command>          - run remote shell command  
  adb emu <command>            - run emulator console command  
  adb logcat [ <filter-spec> ] - View device log  
  adb forward <local> <remote> - forward socket connections  
                                 forward specs are one of:   
                                   tcp:<port>  
                                   localabstract:<unix domain socket name>  
                                   localreserved:<unix domain socket name>  
                                   localfilesystem:<unix domain socket name>  
                                   dev:<character device name>  
                                   jdwp:<process pid> (remote only)  
  adb jdwp                     - list PIDs of processes hosting a JDWP transport  
  adb install [-l] [-r] [-s] <file> - push this package file to the device and install it  
                                 ('-l' means forward-lock the app)  
                                 ('-r' means reinstall the app, keeping its data)  
                                 ('-s' means install on SD card instead of internal storage)  
  adb uninstall [-k] <package> - remove this app package from the device  
                                 ('-k' means keep the data and cache directories)  
  adb bugreport                - return all information from the device  
                                 that should be included in a bug report.  
  
  adb help                     - show this help message  
  adb version                  - show version num  
  
DATAOPTS:  
 (no option)                   - don't touch the data partition  
  -w                           - wipe the data partition  
  -d                           - flash the data partition  
  
scripting:  
  adb wait-for-device          - block until device is online  
  adb start-server             - ensure that there is a server running  
  adb kill-server              - kill the server if it is running  
  adb get-state                - prints: offline | bootloader | device  
  adb get-serialno             - prints: <serial-number>  
  adb status-window            - continuously print device status for a specified device  
  adb remount                  - remounts the /system partition on the device read-write  
  adb reboot [bootloader|recovery] - reboots the device, optionally into the bootloader or recovery program  
  adb reboot-bootloader        - reboots the device into the bootloader  
  adb root                     - restarts the adbd daemon with root permissions  
  adb usb                      - restarts the adbd daemon listening on USB  
  adb tcpip <port>             - restarts the adbd daemon listening on TCP on the specified port  
networking:  
  adb ppp <tty> [parameters]   - Run PPP over USB.  
 Note: you should not automatically start a PPP connection.  
 <tty> refers to the tty for PPP stream. Eg. dev:/dev/omap\_csmi\_tty1  
 [parameters] - Eg. defaultroute debug dump local notty usepeerdns  
  
adb sync notes: adb sync [ <directory> ]  
  <localdir> can be interpreted in several ways:  
  
  - If <directory> is not specified, both /system and /data partitions will be updated.  
  
  - If it is "system" or "data", only the corresponding partition  
    is updated.  
  
environmental variables:  
  ADB\_TRACE                    - Print debug information. A comma separated list of the following values  
                                 1 or all, adb, sockets, packets, rwx, usb, sync, sysdeps, transport, jdwp  
  ANDROID\_SERIAL               - The serial number to connect to. -s takes priority over this if given.  
  ANDROID\_LOG\_TAGS             - When used with the logcat option, only these debug tags are printed.

1) Create AVD (virtual device)

   >> android list targets      // to find out available target id.

     id: 4 or "android-8"  
     Name: Android 2.2  
     Type: Platform  
     API level: 8  
     Revision: 3  
     Skins: WVGA854, WVGA800 (default), QVGA, HVGA, WQVGA400, WQVGA432

>> android create avd -n em22 -t 4 -a

2) Create Project 

>> android create project --name "my app name" \  
                                 --activity "HelloAndroid" \  
                                 --path "~/workplace" \  
                                 --package "com.cs211s.hw1" \  
                                 --target 4   
  
3) Compile/debug  
  
           3.0 You should have a build.xml file to use ant in your project root folder. if not,

     (pjt root)> android update project --path .

   This will create build.xml file.  
  
          3.0  an emulator should be connected before compile or run.

        >> emulator -avd em22   // start the emulator

3.1\_

>> ant compile            // this will create "gen/../R.java" file  
  
    // XML files are also compiled : optimized to the compact size

3.2

>> ant debug              // "debug" means "run"

3.3   You will use this a lot!

>> ant install     // or..  
>> ant clean install         // clean : overwrites the previous one.

     // ant will compile and install the program to the emulator  
     // at this point,  you should have only on device connected (adb devices) or  
     // you should use   
     // adb -s emulator-5554 install bin/HelloAndroid-debug.apk  
     // instead.  
   3.1 -> 3.2 -> 3.3   or  3.1  -> 3.3   or just  3.3   works.

Dividing main.xml

 Instead of making one gigantic main.xml, we can divide this into 'main.xml', 'buttons.xml', 'checkboxes'...etc.

contents of main.xml :   (this code is not from the instructor but from the link above)

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
android:orientation="vertical"  
android:layout\_width="fill\_parent"  
android:layout\_height="fill\_parent"  
>  
<TextView  
android:layout\_width="fill\_parent"  
android:layout\_height="wrap\_content"  
android:text="combining layouts"  
/>  
  
<include android:id="@+id/cell1" layout="@layout/layout2" />  
<include android:id="@+id/cell2" layout="@layout/layout2" />  
<include android:id="@+id/cell3" layout="@layout/layout2" />  
<include android:id="@+id/cell4" layout="@layout/layout2" />

Contents of layout2.xml:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
android:orientation="vertical"  
android:layout\_width="fill\_parent"  
android:layout\_height="100px"  
android:background="#0033cc"  
>  
<TextView  
android:layout\_width="fill\_parent"  
android:layout\_height="40px"  
android:text="layout2"  
/>  
<CheckBox  
android:layout\_width="fill\_parent"  
android:layout\_height="40px"  
/>  
</LinearLayout>

1) what activity has invoked this activity?  
  
   this.getIntent();  
  
        Context ct=((Context) getApplication());  
        Intent i =getIntent();  
        if(i==null){  // this activity is a launcher activity  
                  // or go to the manifest file look at launcher  
        // how many activity files are going on? without seeing manifest file  
        // >>  aapt -list -v xxx.apk  will show the activities  
        // what will makes java file an activity: which exends Activity  
    // say.. public class Authen extends Activity

1) lcat : shell script  
   lcat "regEX"  
  \*lcat is not working if the emulatoris not running.   
  
2) adb logcat -c // clear the logcat

3) checking emulator  
  
   if(System.Enviroment.DeviceType != DeviceType.Emulator)  
    Log.e(TAG,"emulator is not working!");  
  
4) how to check my 'app' is running either on a device or an emulator ?  
  
   if("google.sdk".equals(Build.PRODUCT)  
    Log.e(TAG," is running on the device");  
  
5) String android\_id = secure.getString( getContentResolver(),  
            Secure.ANDROID\_ID);  
   if(android\_id== null)  
    Log.e(TAG,"running on the emulator ");  
   else  
      
    Log.e(TAG,"running on the device ");  
  
  
6)  
   public boolean isEmulator(){ return (BUILD.MANUFACTURER.equals("unknown")}  
      
   [shell]    
    a=$(adb devices|grep emulator)  
    if [ -z "$a" ]  
        echo "no emulator is up or running"  
        exit 1  
    fi  
  
    b=$(echo $a | grep "offline")  
    if [ -n "$b" ]  
        echo "no emulator is up or running"  
        exit 1  
    fi

#1. what is the difference between

   bin/doit.ap\_          :  compiled file of resources

   /bin/doit-xxx.apk   :  final compiled file

                 this file contains the android application's code and resources, .jar files. And this file gets installed onto device.

#2. Two jobs of aapt. :

   1) aapk compiles all the resources except raw resources and places them all into the final .apk file.

   2) creates R.java

#3. without looking at R.java. how would you know about names of inner classes in R.java?

   julie@ubuntu:~/workspace/Tempconverter$ ls bin/com/hw4/ | grep 'R\$'

4.How to get the version of Android

(V1.5)  Andriod.os.Build.VERSION.SDK

(V1.6 or higher)  Andriod.os.Build.VERSION.SDK\_INT

        device.setText("android.os.Build.DEVICE: " + android.os.Build.DEVICE);

        model.setText("android.os.Build.MODEL: " + android.os.Build.MODEL);

        product.setText("android.os.Build.PRODUCT: " + android.os.Build.PRODUCT);

        codename.setText("android.os.Build.VERSION.CODENAME: " + android.os.Build.VERSION.CODENAME);

        incremental.setText("android.os.Build.VERSION.INCREMENTAL: " + android.os.Build.VERSION.INCREMENTAL);

        release.setText("android.os.Build.VERSION.RELEASE: " + android.os.Build.VERSION.RELEASE);

        sdk.setText("android.os.Build.VERSION.SDK: " + android.os.Build.VERSION.SDK);

        sdkInt.setText("android.os.Build.VERSION.SDK\_INT: " + String.valueOf(android.os.Build.VERSION.SDK\_INT));

String versionName = getPackageManager().getPackageInfo(getPackageName(),0).versionName;  
int versionCode = getPackageManager().getPackageInfo(getPackageName(),0).versionCode;  
Log.i(TAG,Android.os.Build.VERSION\_CODE.FROYO);

Log.i(TAG,Android.os.Build.VERSION\_CODE.DONUT);

 bin/classes/<package>

ID

you refer to Android resources , which are already defined in Android system, with@android:id/.. while to access resources that you have defined/created in your project, you use@id/..

it depends on the context, when you're using the XML attribute of android:id, then you're specifying a new id, and are instructing the parser (or call it the builder) to create a new entry inR.java, thus you have to include a + sign.

While in the other case, like android:layout\_below="@id/myTextView" , you're referring to an id that has already been created, so parser links this to the already created id in R.java

The '+' means to create the symbol if it doesn't already exist.

Toast

For example, if you decide that the toast should appear in the top-left corner, you can set the gravity like this:

String android\_id = Secure.getString(getContentResolver(), Secure.ANDROID\_ID);

String tag = "WHERE";

if(android\_id == null)

{

Log.i(tag, "We are in emulator");

}

Log.i(tag, android\_id);

Log.i(tag, "NEXT CHECK ");

if("google\_sdk".equals(Build.PRODUCT))

{

Log.i(tag, "ON PHONE");

}

else

Log.i(tag, android\_id);

Log.i(tag, "NEXT CHECK ");

if("sdk".equals(Build.PRODUCT))

{

Log.i(tag, "ON PHONE");

}

else

Log.i(tag, android\_id);

Log.i(tag, "NEXT CHECK ");

if(Build.MANUFACTURER.equals("unknown")) \*\*\*\*\*\*\*\*\*\*\*

{

Log.i(tag, "EMULATOR");

}

else

Log.i(tag, android\_id);

Log.i(tag, "Get android version");

Log.i(tag, "" + Build.VERSION.SDK\_INT);

String versionName;

int version;

try

{

versionName = getPackageManager().getPackageInfo(getPackageName(), 0).versionName;

Log.i(tag, versionName);

version = getPackageManager().getPackageInfo(getPackageName(), 0).versionCode;

Log.i(tag, "" + version);

} catch (NameNotFoundException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

String v = android.os.Build.VERSION.CODENAME;

int i = android.os.Build.VERSION\_CODES.DONUT;

Log.i(tag, v);

Log.i(tag, "" +i);

int s = android.os.Build.VERSION.SDK\_INT;

if(this.getIntent().getData() == null)

Log.i(tag, "I AM THE LAUNCHER");

------ Using System.out/err -----------

 DVM dumps/redirect all the System.out(err) to /dev/null  
  
  >> adb shell stop  
  >> adb shell setprop log.    
  >>     # will be reset to false when restart the emulator  
  >> adb shell start  
  
   (1) by editing local.prop  
  \* lookup the file /data/local.prop  
  \* add this line  
    redirect-stdio=true    
  
   >> adb shell getprop log.redirect-stdio  
   >> true  
  
   (2) by java  
   System.setProperty("log.redirect-stdio=true");  
   System.out.println(" will work");

public boolean isEmulatorUp()  
{  
   return (Build.MANUFACTURER.equalsIgnoreCase("unknown"));  
}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** 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);

    }