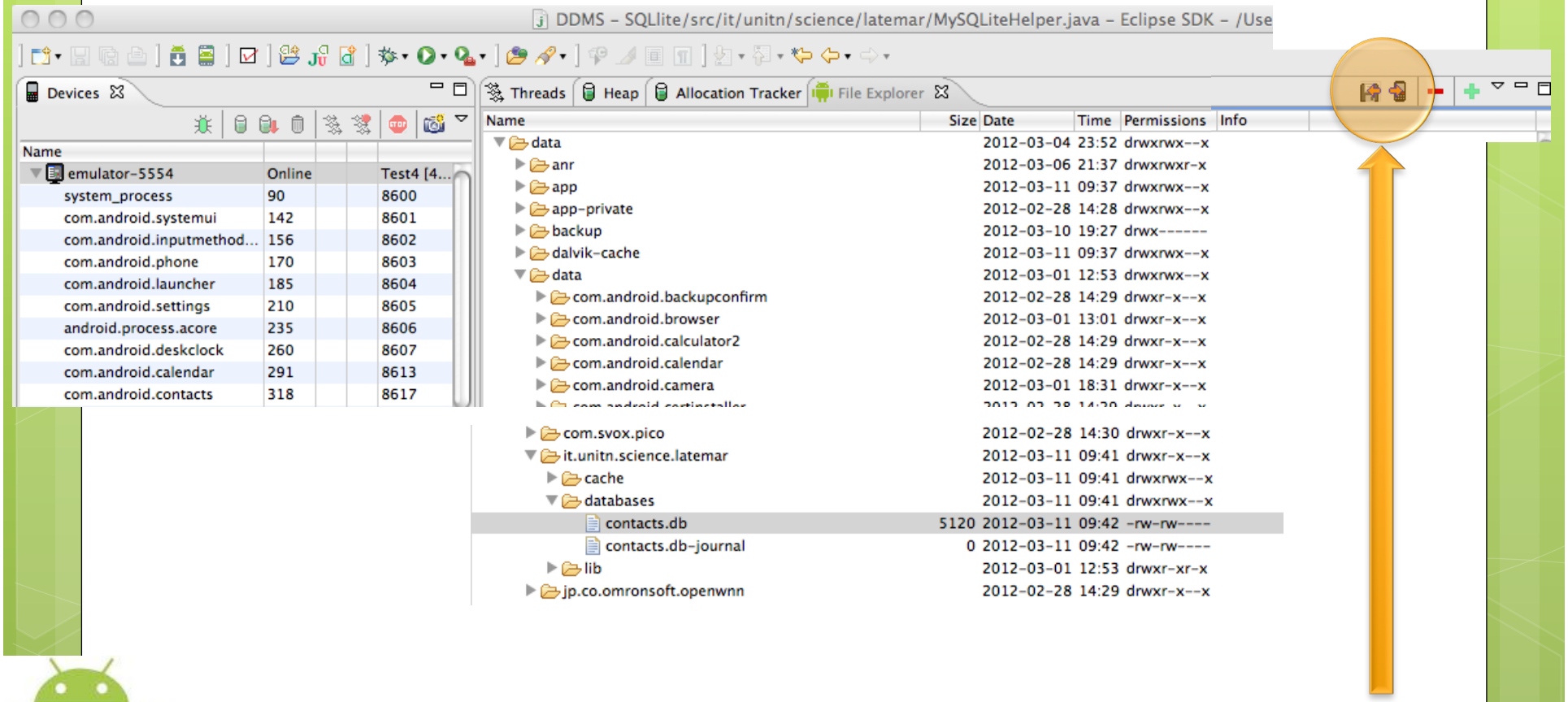
How to access your database from the development environment

Marco Ronchetti
Università degli Studi di Trento

Find your DB

1) Look into data/data/YOURPACKAGE/databases/YOURDATABASE.db



2) Pull the file on your PC

3) Use sqlite on your PC (in your_sdk_dir/tools)

Access your DB

Use the following script, and

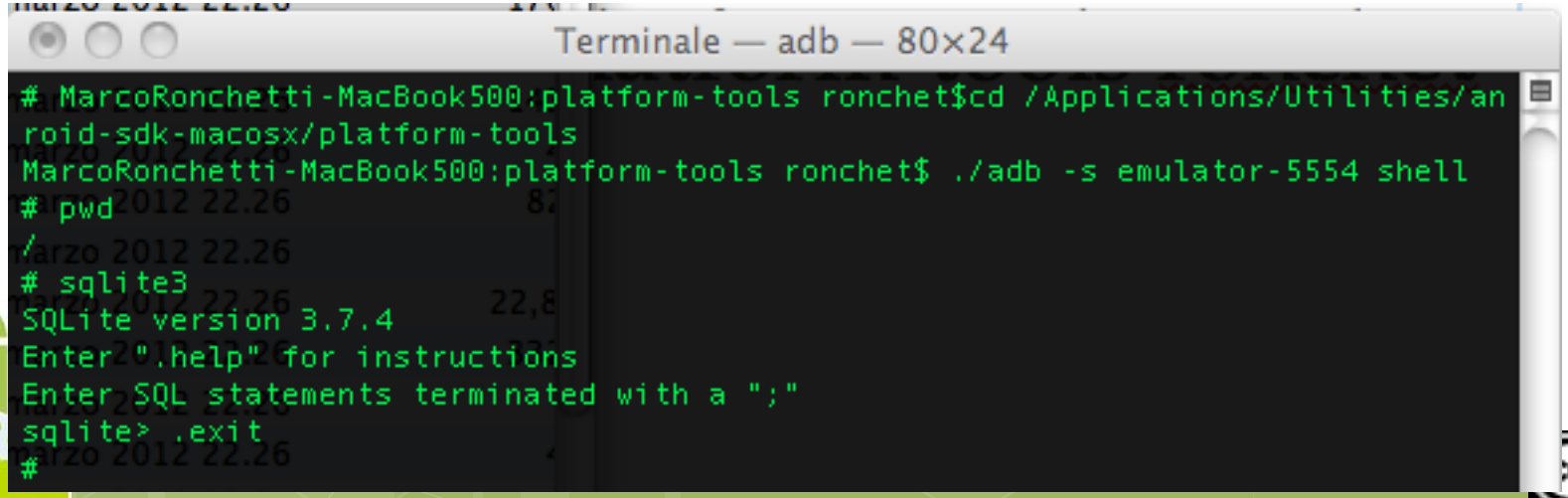
```
#!sh
adb shell "chmod 777 /data/data/com.mypackage/databases/store.db"
adb pull /data/data/com.mypackage/databases/store.db
```

OR

Run remote shell

```
$ adb –s emulator-5554 shell
$ cd /data/data/com.yourpackage/databases
$ sqlite3 your-db-file.db
> .help
```

adb -s <serialNumber> <command> to access a device



adb

adb is in your android-sdk/platform-tools directory

It allows you to:

- Run shell commands on an emulator or device
- Copy files to/from an emulator or device
- Manage the state of an emulator or device
- Manage port forwarding on an emulator or device

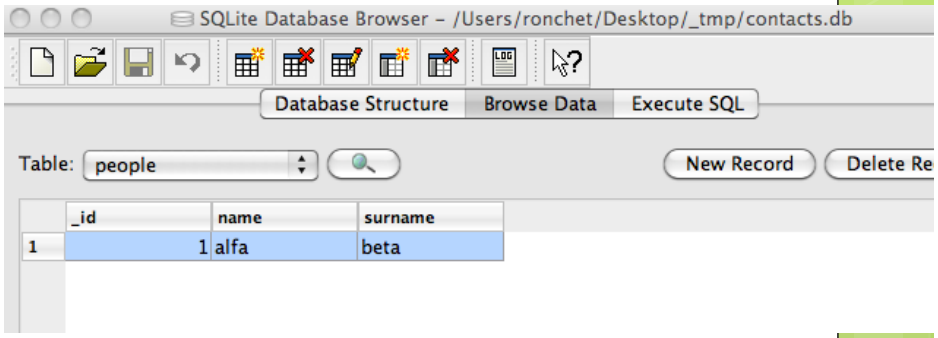
It is a client-server program that includes three components:

- A client, which runs on your development machine.
- A daemon, which runs as a background process on each emulator or device instance.
- A server, which runs as a background process on your development machine and manages communication between the client and the daemon.

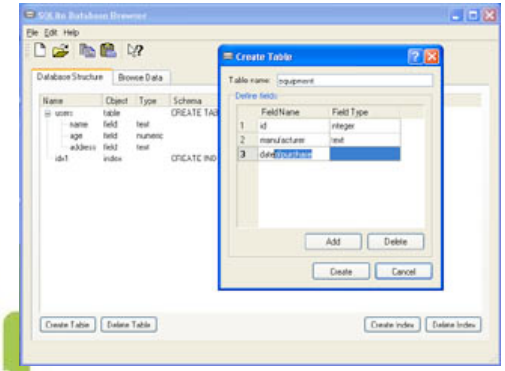
See http://developer.android.com/tools/help/adb.html

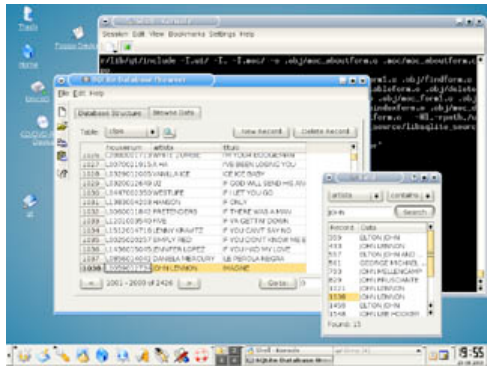
A graphical sqlite browser

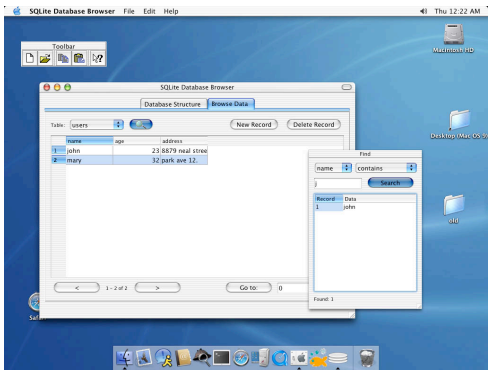
http://sqlitebrowser.org/



- Create and compact database files
- Create, define, modify and delete tables
- Create, define and delete indexes
- Browse, edit, add and delete records
- Search records
- Import and export records as text
- Import and export tables from/to CSV files
- Import and export databases from/to SQL dump files
- Issue SQL queries and inspect the results
- Examine a log of all SQL commands issued by the application







6

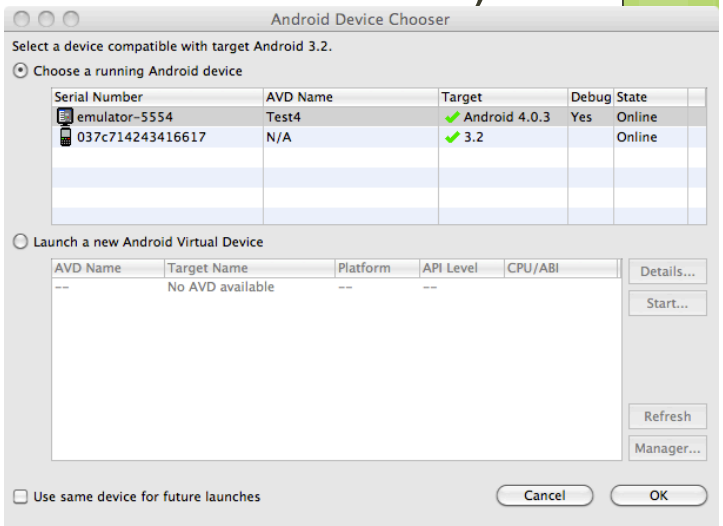
Testing and deploying on your device

Marco Ronchetti
Università degli Studi di Trento

Configure device

1) Turn on "USB Debugging" on your device.
   On the device, go in
   - Android <4: **Settings > Applications > Development**
   - Android>=4: **Settings > Developer options**

   and enable **USB debugging**
2) Load driver on PC (win-linux, on Mac not needed)
3) Check in shell: adb devices
4) In Eclipse, you'll have the choice



Make sure the version of OS is correct both in project properties
And in manifest!

See http://developer.android.com/guide/developing/device.html

Alternative, simple way to deploy

e.g. to give your app to your friends

Get Dropbox both on PC and Android device
Copy your apk from bin/res into dropbox (on PC)
Open dropbox on Android device, and open your apk

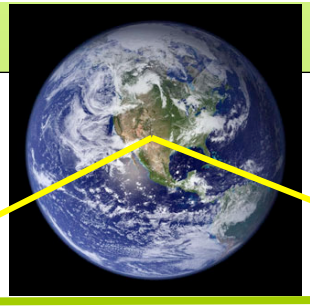
By sharing your dropbox with others you can easily pass your app.

www.dropbox.com

DAO Implementation
File System

Marco Ronchetti
Università degli Studi di Trento

ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DB

Actual storage

FS

platforms

temp

tools

Data

Object

The Java-IO philosophy

1) Get a (raw) source

File f; … ; InputStream s = new FileInputStream(f);
Socket s; … ; InputStream s=s.getInputStream();
StringBuffer b; … ; InputStream s = new StringBufferInputStream(f);

2) Add functionality

Reader r=new InputStringReader(s); //bridge class
DataInputString dis=new DataInputString(s); //primitive data
ObjectInputString ois=new ObjectInputString(s); //serialized objects

3) Compose multiple functionalities

```
InputStream es=new FilteredInputStream(
                    new BufferedInputStream(
                    new PushBacklputStream(s)));
```

Choose the type of source!

You can choose among four types of basic sources:

	BYTE		CHARACTER	
SOURCE	InputStream	OutputStream	Reader	Writer

Both file and directory information is available via the File class, or the classes (like Path) in the nio package.

I/O Table

	Byte Based		Character Based	
	Input	*Output*	*Input*	*Output*
Basic	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream RandomAccessFile	FileOutputStream RandomAccessFile	FileReader	FileWriter
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream StreamTokenizer		PushbackReader LineNumberReader	
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

Android internal file I/O

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

FileOutputStream fos = openFileOutput(FILENAME,
        Context.MODE_PRIVATE); // called in a Context

fos.write(string.getBytes());
fos.close();
```

Using temporary files

```
File file = new File(getCacheDir(), "temp.txt");
 try {
    file.createNewFile();
    FileWriter fw = new FileWriter(file);
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("Hello World\n");
    bw.close();
} catch (IOException e) {
        Toast.makeText(this,
            "Error creating a file!”
            ,Toast.LENGTH_SHORT).show();
}
```

When the device is low on internal storage space, Android may delete these cache files to recover space.

You should not rely on the system to clean up these files for you.

Clean the cache files yourself

stay within a reasonable limit of space consumed, such as 1MB.

Other useful methods

getFilesDir()

Get the absolute path where internal files are saved.

getDir()

Creates (or opens an existing) directory within your internal storage space.

deleteFile()

Deletes a file saved on the internal storage.

fileList()

Returns an array of files currently saved by your application.

The DAO interface

```
package it.unitn.science.latemar;

import java.util.List;

public interface PersonDAO {
        public void open();
        public void close();

        public Person insertPerson(Person person) ;
        public void deletePerson(Person person) ;
        public List<Person> getAllPerson() ;
}
```

package it.unitn.science.latemar;
import ...

The DAO implementation - FS

```
public class PersonDAO_FS_impl implements PersonDAO {
        DataOutputStream fos;
        DataInputStream fis;
        Context context=MyApplication.getAppContext();
        final String FILENAME="contacts";

@Override
        public void open() {
           try {
               fos=new DataOutputStream(
                     context.openFileOutput(FILENAME, Context.MODE_APPEND)
               );
           } catch (FileNotFoundException e) {e.printStackTrace();}

        }
@Override
        public void close() {
           try {
                fos.close();
            } catch (IOException e) {e.printStackTrace();}
        }
```

This should never happen

The DAO impl. – data access 2

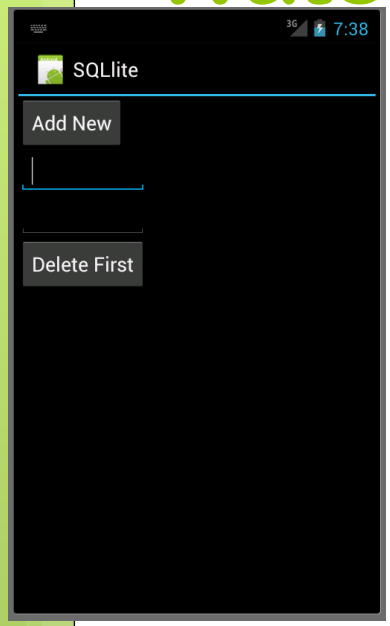
```
@Override
   public Person insertPerson(Person person) {
        try {
            fos.writeUTF(person.getName());
            fos.writeUTF(person.getSurname());
        } catch (IOException e) { e.printStackTrace(); }
        return person;
    }

@Override
    public void deletePerson(Person person) {
        Log.d("trace","deletePerson DAO_FS – UNIMPLEMENTED!");
     }
```

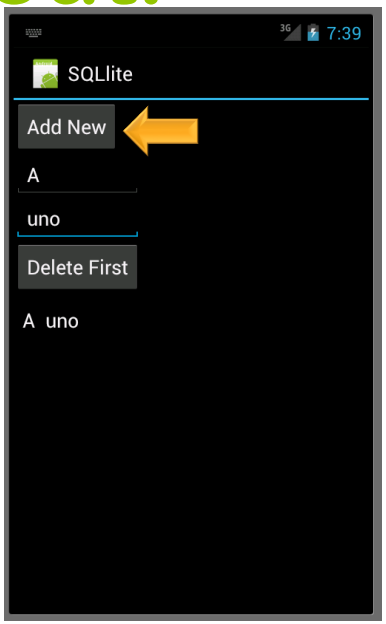
write as Unicode

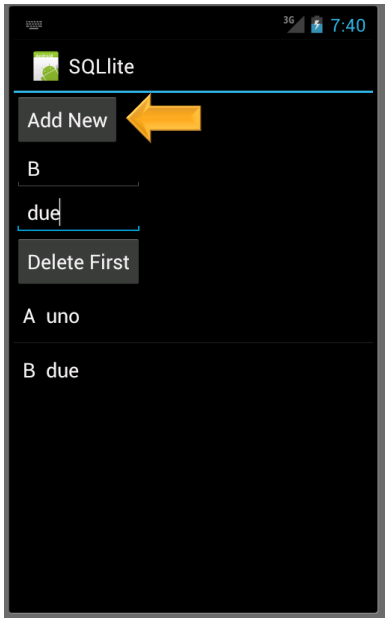
The DAO impl. – data access 3

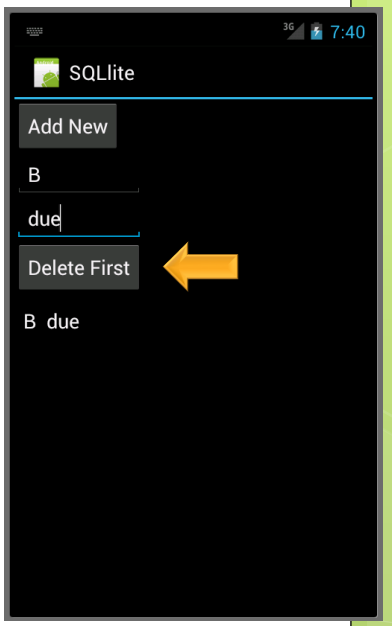
```
@Override
   public List<Person> getAllPersons() {
       List<Person> list=new ArrayList<Person>();
       try { fis=new DataInputStream( context.openFileInput(FILENAME) );
        } catch (FileNotFoundException e) {
            e.printStackTrace();    return list;
        }
        while (true) {
            try {
                    String name=fis.readUTF();
                    String surname=fis.readUTF();
                    Person p=new Person(name, surname);
                    list.add(p);
             } catch (EOFException e) {  break;
             } catch (IOException e) { e.printStackTrace(); break; }
      }
      try { fis.close(); } catch (IOException e) { e.printStackTrace(); }
      return list;
   }
```

Watch out!

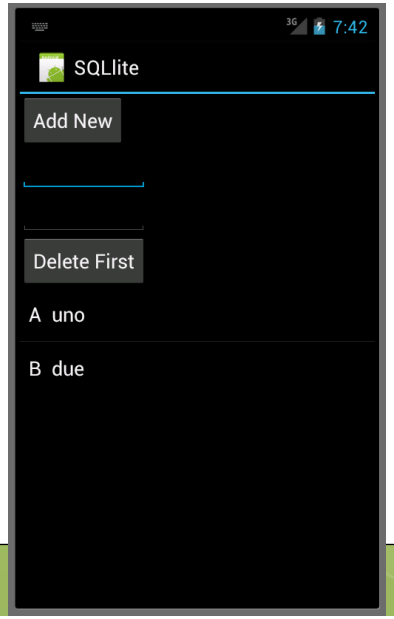








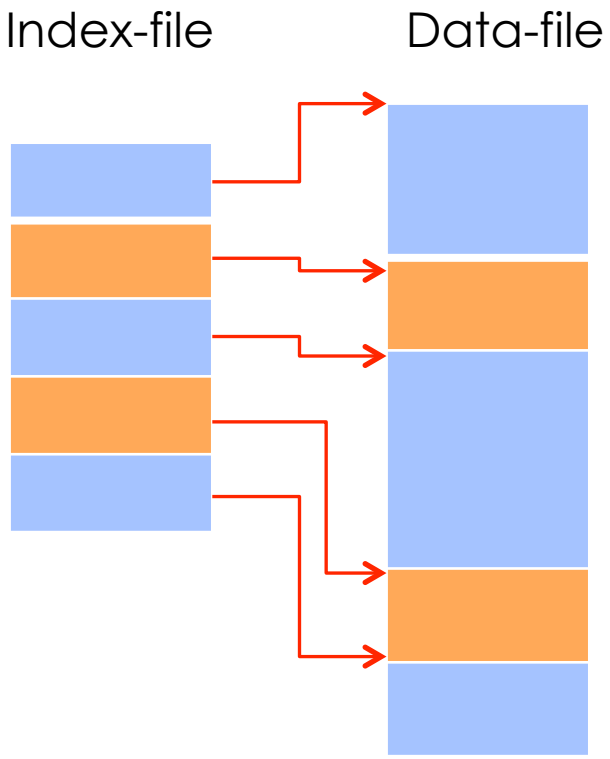
Restart…



Why so?

Serializing any-size objects to a random access file

http://www.maridonkers.info/serializing-any-size-objects-to-a-random-access-file-2/



See iava.io
Class RandomAccessFile

External Files

Marco Ronchetti
Università degli Studi di Trento

External storage

Every Android-compatible device supports a shared "external storage" that you can use to save files.

It can be:

- a removable storage media (such as an SD card)
- an internal (non-removable) storage.

Files saved to the external storage

- are world-readable
- can be modified by the user when the USB card storage in moved on a computer!

Possible states of external media

String Environment.getExternalStorageState();

MEDIA_MOUNTED
- media is present and mounted at its mount point with read/write access.

MEDIA_MOUNTED_READ_ONLY
- media is present and mounted at its mount point with read only access.

MEDIA_NOFS
- media is present but is blank or is using an unsupported filesystem

MEDIA_CHECKING
- media is present and being disk-checked

MEDIA_UNMOUNTED
- media is present but not mounted

MEDIA_SHARED
- media is in SD card slot, unmounted, and shared as a mass storage device.

MEDIA_UNMOUNTABLE
- media is present but cannot be mounted.

MEDIA_REMOVED
- media is not present.

boolean Environment.isExternalStorageEmulated()
boolean Environment.isExternalStorageRemovable()

MEDIA_BAD_REMOVAL
- media was removed before it was unmounted.

Standard directories (constants):

DIRECTORY_DOWNLOADS
- files that have been downloaded by the user.

DIRECTORY_MOVIES
- movies that are available to the user.

DIRECTORY_PICTURES
- pictures that are available to the user.

DIRECTORY_DCIM
- The traditional location for pictures and videos when mounting the device as a camera.

Places for audio files:
- DIRECTORY_MUSIC
  - music for the user.
- DIRECTORY_ALARMS
  - alarms sounds that the user can select (not as regular music).
- DIRECTORY_NOTIFICATIONS
  - notifications sounds that the user can select (not as regular music).
- DIRECTORY_PODCASTS
  - podcasts that the user can select (not as regular music).
- DIRECTORY_RINGTONES
  - ringtones that the user can select (not as regular music).

Other Environment static methods

static File **getRootDirectory**()

- Gets the Android root directory (typically returns /system).

static File **getDataDirectory**()

- Gets the Android data directory (typically returns /data).

static File **getDownloadCacheDirectory**()

- Gets the Android Download/Cache content directory. Here go **temporary** files that **are specific to your application** If the user uninstalls your application, this directory and all its contents will be deleted. You should manage these cache files and remove those that aren't needed in order to preserve file space.

static File **getExternalStorageDirectory**()

- Gets the Android external storage directory. Here go files that **are specific to your application** If the user uninstalls your application, this directory and all its contents will be deleted.

static File **getExternalStoragePublicDirectory**(String type)

- Get a top-level public external storage directory for placing files of a particular type. This is where the user will typically place and manage their own files. Here go **files that are not specific to your application** and that should *not* be deleted when your application is uninstalled

Rooting a device

Marco Ronchetti
Università degli Studi di Trento

Rooting

The process of allowing users of Android devices to get root access. Varies widely by device, as it usually exploits a security weakness in the firmware shipped from the factory.

Goal:

- to overcome limitations imposed by that carriers and hardware manufacturers
- to alter or replace system applications and settings
- to run specialized apps that require administrator-level permissions
- to perform other operations that are otherwise inaccessible to a normal Android user.

The process of rooting

On the iphone: jailbreaking

e.g.: CyanogenMod

a replacement firmware. Offers several features, like:

- an OpenVPN client,
- a reboot menu,
- CPU overclocking and performance enhancements, app permissions management

Over 1.5 M installations

Is it legal?

On July 26, 2010, the U.S. Copyright office announced a new exemption making it officially legal to root a device and run unauthorized third-party applications, as well as the ability to unlock any cell phone for use on multiple carriers.

Industry reaction

- concern about improper functioning of devices running unofficial software and related support costs.
- offers features for which carriers would otherwise charge a premium

Technical obstacles have been introduced in many devices (e.g. locked bootloaders).

In 2011 an increasing number of devices shipped with unlocked or unlockable bootloaders.

The HTC case

"HTC is committed to listening to users and delivering customer satisfaction. We have heard your voice and starting now, we will allow our bootloader to be unlocked for 2011 models going forward.

It is our responsibility to caution you that not all claims resulting or caused by or from the unlocking of the bootloader may be covered under warranty.

We strongly suggest that you do not unlock the bootloader unless you are confident that you understand the risks involved."

See e.g. http://htcdev.com/bootloader/

Fragments

Fragments

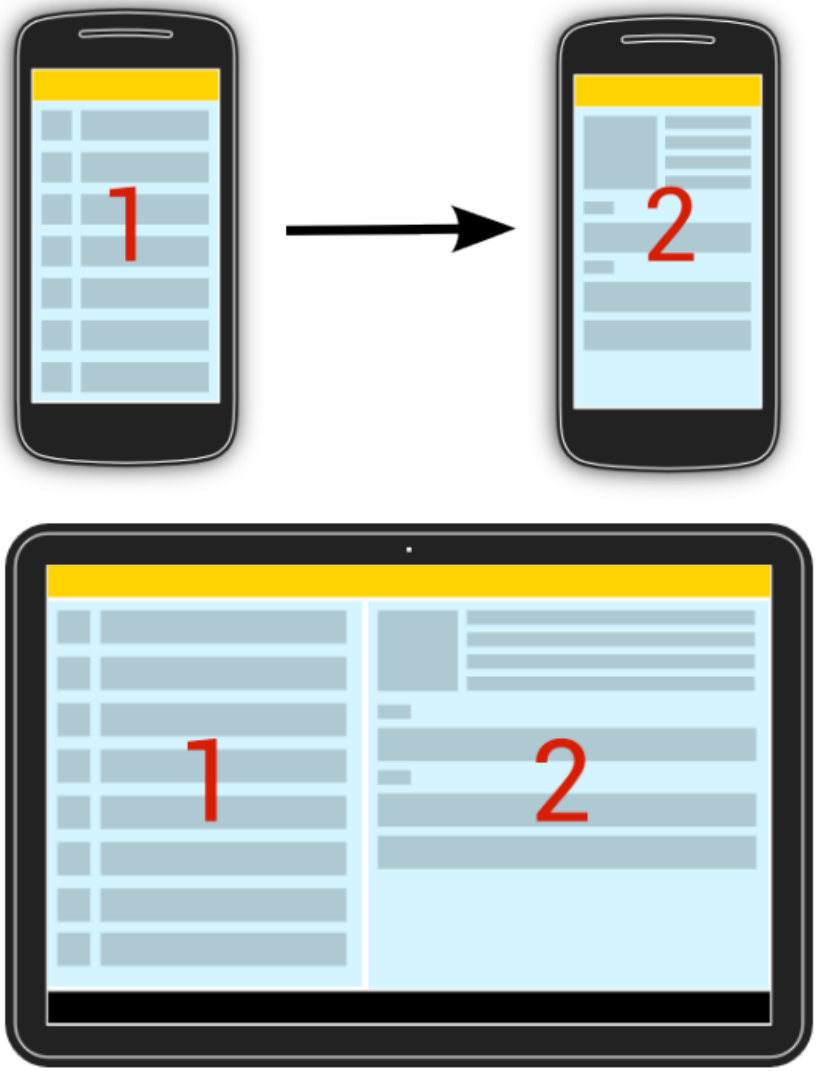
A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity.

Fragments can be assembled to create an activity during the application design phase, and added to, or removed from an activity during application runtime to create a dynamically changing user interface.

Fragments may only be used as part of an activity and cannot be instantiated as standalone application elements.

A fragment can be thought of as a functional "sub-activity" with its own lifecycle similar to that of a full activity.

Using fragments



Fragments lifecycle

Method	Description
onAttach()	The fragment instance is associated with an activity instance.The activity is not yet fully initialized
onCreate()	Fragment is created
onCreateView()	The fragment instance creates its view hierarchy. The inflated views become part of the view hierarchy of its containing activity.
onActivityCreated()	Activity and fragment instance have been created as well as thier view hierarchy. At this point, view can be accessed with the `findViewById()` method. example.
onResume()	Fragment becomes visible and active.
onPause()	Fragment is visibile but becomes not active anymore, e.g., if another activity is animating on top of the activity which contains the fragment.
onStop()	Fragment becomes not visible.

Defining a new fragment (from code)

To define a new fragment you either extend the android.app.Fragment class or one of its subclasses, for example, ListFragment, DialogFragment, PreferenceFragment or WebViewFragment.

Defining a new fragment (from code)

```
public class DetailFragment extends Fragment {
 @Override
 public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
    View view=inflater.inflate(
        R.layout.fragment_rssitem_detail,
        container, false);
   return view;
 }
 public void setText(String item) {
    TextView view = (TextView)
                getView().findViewById(R.id.detailsText);
    view.setText(item);
 }
}
```

XML-based fragments

```
<RelativeLayout xmlns:android="http://schemas.android.com/
apk/res/android" xmlns:tools="http://schemas.android.com/
tools" android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".FragmentDemoActivity" >
<fragment android:id="@+id/fragment_one"
android:name="com.example.myfragmentdemo.FragmentOne"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_centerVertical="true" tools:layout="@layout/
fragment_one_layout" />
</RelativeLayout>
```

Adding-removing fragments at runtime

The FragmentManager class and the FragmentTransaction class allow you to add, remove and replace fragments in the layout of your *activity*.

Fragments can be dynamically modified via transactions. To dynamically add fragments to an existing layout you typically define a container in the XML layout file in which you add a *Fragment*.

```
FragmentTransaction ft =
getFragmentManager().beginTransaction();
ft.replace(R.id.your_placehodler, new
YourFragment());
ft.commit();
```

A new *Fragment* will replace an existing *Fragment* that was previously added to the container.

Finding if a fragment is already part of your Activity

```
DetailFragment fragment = (DetailFragment)
        getFragmentManager().
        findFragmentById(R.id.detail_frag);

if (fragment==null) {
                // start new Activity
} else {
                fragment.update(...);
}
```

Communication: activity -> fragment

In order for an activity to communicate with a fragment, the activity must identify the fragment object via the ID assigned to it using the findViewById() method. Once this reference has been obtained, the activity can simply call the public methods of the fragment object.

Communication: fragment-> activity

Communicating in the other direction (from fragment to activity) is a little more complicated.

A) the fragment must define a listener interface, which is then implemented within the activity class.

```
public class MyFragment extends Fragment {
  AListener activityCallback;
  public interface AListener {
            public void someMethod(int par1, String par2);
  }
  …
```

Communication: fragment-> activity

B. the onAttach() method of the fragment class needs to be overridden and implemented. The method is passed a reference to the activity in which the fragment is contained. The method must store a local reference to this activity and verify that it implements the interface.

```
public void onAttach(Activity activity) {
  super.onAttach(activity);
  try { activityCallback = (AListener) activity;
  } catch (ClassCastException e) {
      throw new ClassCastException(
            activity.toString()
            + " must implement ToolbarListener");
} }
```

Communication: fragment-> activity

C. The next step is to call the callback method of the activity from within the fragment. When and how this happens is entirely dependent on the circumstances under which the activity needs to be contacted by the fragment. For the sake of an example, the following code calls the callback method on the activity when a button is clicked:

```
public void buttonClicked(View view) {
    activityCallback.someMethod(arg1, arg2);
}
```

Communication: fragment-> activity

All that remains is to modify the activity class so that it implements the ToolbarListener interface.

```
public class MyActivity extends
FragmentActivity implements
MyFragment.AListener {
 public void someMethod(String arg1, int arg2)
      {
      // Implement code for callback method
      }
.
.
}
```

Esempio

vedi
http://www.vogella.com/tutorials/AndroidFragments/article.html

sez. 10