Preferences

Marco Ronchetti
Università degli Studi di Trento

SharedPreferences

SharedPreferences allows to save and retrieve persistent key-value pairs of primitive data types. This data will persist across user sessions (even if your application is killed).

getSharedPreferences(String name, int mode)

A method of Contex

- Uses multiple preferences files identified by name, which you specify with the first parameter.

getPreferences()

A method of Activity

- Use this if you need only one preferences file for your Activity. This simply calls the underlying getSharedPreferences(String, int) method by passing in this activity's class name as the preferences name

SharedPreferences methods

boolean contains(String key)

Checks whether the preferences contains a preference.

Value returned
If key does not exist

T getT(String key, T defValue)

Retrieve a T value from the preferences where T={int, float, boolean, long, String, Set<String>}.

SharedPreferences.Editor edit()

All changes you make in an editor are batched, and not copied back to the original SharedPreferences until you call commit() or apply()

SharedPreferences.Editor methods

Void apply(), boolean commit()
Commit your preferences changes back (apply is asynchronous)

Editor putT(String key, T value)
Stores a T value in the preferences where T={int, float, boolean, long, String, Set<String>}.

Editor remove(String key)
Mark in the editor that a preference value should be removed

Editor clear ()
Mark in the editor that all preference values should be removed

User Preferences

Shared preferences are not strictly for saving "user preferences," such as what ringtone a user has chosen.

For creating user preferences for your application, you should use PreferenceActivity, which provides an Activity framework for you to create user preferences, which will be automatically persisted (using shared preferences).

It is based on Fragments

Threads

Marco Ronchetti
Università degli Studi di Trento

Threads

When an application is launched, the system creates a thread of execution for the application, called "main" or "UI thread"
This thread dispatches events to the user interface widgets, and draws (uses the android.widget and android.view packages).

Unlike Java AWT/Swing, separate threads are NOT created automatically.
Methods that respond to system callbacks (such as onKeyDown() to report user actions or a lifecycle callback method) always run in the UI thread.

If everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang.

If the UI thread is blocked for more than 5 sec the user is presented with the"ANR - application not responding" dialog.

the Andoid UI toolkit is not thread-safe !

Consequence:

you must not manipulate your UI from a worker thread—all manipulation to the user interface must be done within the UI thread.

You MUST respect these rules:

- Do not block the UI thread
- Do not access the Android UI toolkit from outside the UI thread

An example from android developers

```
public void onClick(View v) {
        Bitmap b = loadImageFromNetwork(
                        "http://example.com/image.png");
        myImageView.setImageBitmap(b);
}
```

WRONG! Potentially Slow Operation!

```
public void onClick(View v) {
  new Thread(new Runnable() {
      public void run() {
          Bitmap b = loadImageFromNetwork(
              "http://example.com/image.png");
          myImageView.setImageBitmap(b);
      }})
              .start();
}
```

WRONG! A non UI thread accesses the UI!

Still not the solution…

```
public void onClick(View v) {
  Bitmap b;
      new Thread(new Runnable() {
          public void run() {
              b = loadImageFromNetwork(
                  "http://example.com/image.png");
          }})
      .start();
      myImageView.setImageBitmap(b);
}
```

WRONG!
This does not wait for the thread to finish!

The solution

public boolean post (Runnable action)

- Causes the Runnable to be sent to the UI thread and to be run therein. It is invoked on a View from outside of the UI thread.

public boolean postDelayed (Runnable action, long delayMillis)

```
public void onClick(View v) {
  new Thread(new Runnable() {
      public void run() {
          Bitmap b = loadImageFromNetwork(
              "http://example.com/image.png");
          myImageView.post(
              new Runnable() {
                  public void run() {
                      myImageView.setImageBitmap(bitmap);
                  }
              }
      )}})
          .start();
}
```

OK! This code will be run in the UI thread

Java reminder: varargs

void f(String pattern, Object... arguments);

The three periods after the final parameter's type indicate that the final argument may be passed

- as an array *or*
- as a sequence of arguments.

Varargs can be used *only* in the final argument position.

```
Object a, b, c, d[10];
...
f("hello",d);
f("hello",a,b,c);
```

Varargs example

```
public class Test {
    public static void main(String args[]){ new Test(); }

    Test(){
        String k[]={"uno","due","tre"};
        f("hello",k);
        f("hello",“alpha“,“beta“);
        // f("hello“,“alpha“,“beta“,k); THIS DOES NOT WORK!
    }

    void f(String s, String... d){
        System.out.println(d.length);
         for (String k:d) {
             System.out.println(k);
         }
    }

}
```

AsyncTask<Params,Progress,Result>

Creates a new asynchronous task. The constructor must be invoked on the UI thread.

AsyncTask must be subclassed, and instantiated in the UI thread.

Methods to be overridden:

method	where	when
void onPreExecute()	UI Thread	before
Result doInBackground(Params...)	Separate new thread	during
void onProgressUpdate(Progress...)	UI Thread	
void onPostExecute(Result)	UI Thread	after

The more elegant solution

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://example.com/image.png");
}
```

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }
    protected void onPostExecute(Bitmap result) {
        myImageView.setImageBitmap(result);
    }
}
```

Using Progress

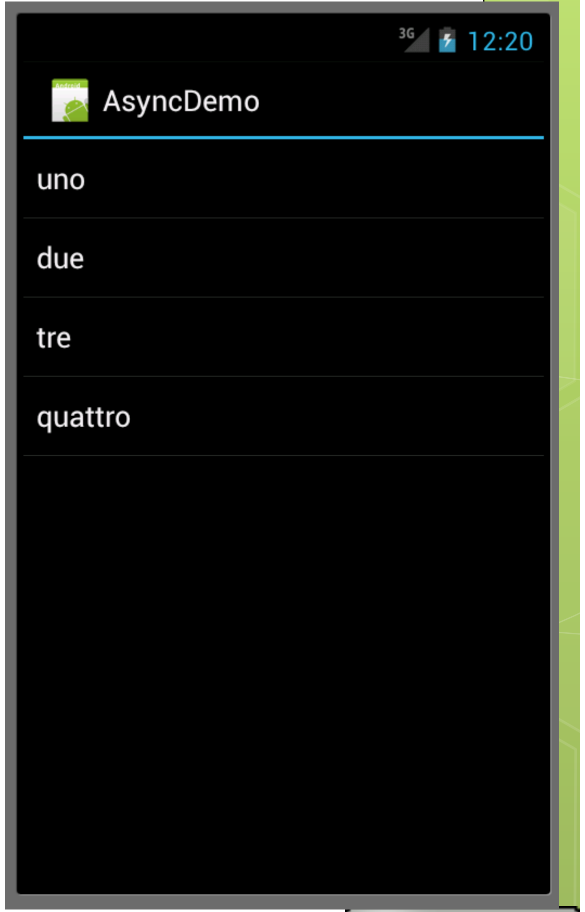
```
package it.unitn.science.latemar;
import …
```

```
public class AsyncDemoActivity extends ListActivity {
 private static final String[] item{"uno","due","tre","quattro",
            "cinque","sei”, "sette","otto","nove",
            "dieci","undici","dodici",};

 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  ListView listView = getListView();

  setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1,
            new ArrayList<String>()));

  new AddStringTask().execute();
 }
```



Adapted from the source code of http://commonsware.com/Android/

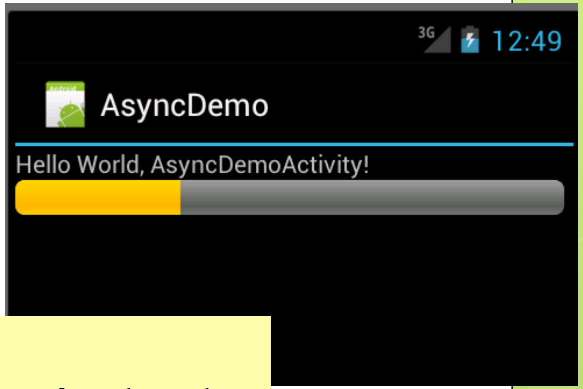
Using Progress

This is an inner class!

```
class AddStringTask extends AsyncTask<Void, String, Void> {
  @Override
  protected Void doInBackground(Void... unused) {
    for (String item : items) {
      publishProgress(item);
      SystemClock.sleep(1000);
    }
    return(null);
  }
  @SuppressWarnings("unchecked")
  @Override
  protected void onProgressUpdate(String... item) {
    ((ArrayAdapter<String>)getListAdapter()).add(item[0]);
  }
```

```
@Override
    protected void onPostExecute(Void unused) {
      Toast
        .makeText(AsyncDemoActivity.this,
              "Done!", Toast.LENGTH_SHORT)
        .show();
    }
  }
}
```

Using the ProgressBar



Hello World, AsyncDemoActivity!

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello" />
   <ProgressBar
       android:id="@+id/pb1"
       android:max="10"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       style="@android:style/Widget.ProgressBar.Horizontal"
       android:layout_marginRight="5dp" />
</LinearLayout>
```

```
public class AsyncDemoActivity2
                extends Activity {
 ProgressBar pb;
 @Override
 public void onCreate(Bundle state) {
  super.onCreate(state);
  setContentView(R.layout.main);
  pb=(ProgressBar) findViewById(R.id.pb1);
  new AddStringTask().execute();
 }
```

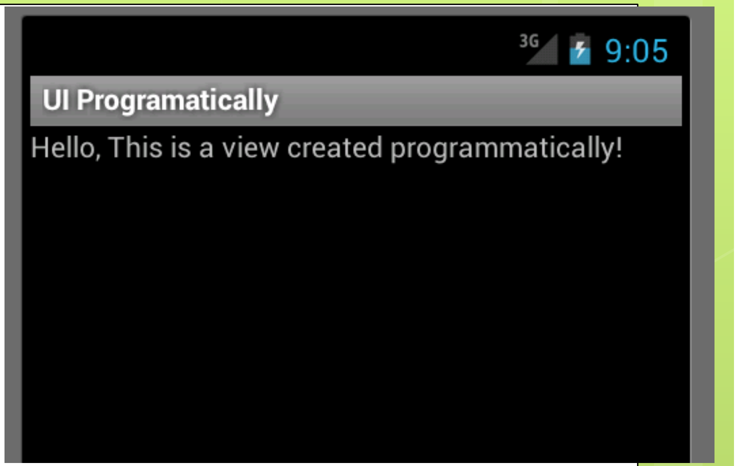
Using the ProgressBar

```
class AddStringTask extends AsyncTask<Void, Integer, Void> {
@Override
protected void doInBackground(Void... unused) {
  int item=0;
  while (item<10 ){
    publishProgress(++item);
    SystemClock.sleep(1000);
  }
}
@Override
protected void onProgressUpdate(Integer... item) {
  pb.setProgress(item[0]);
}
}
```

Basic UI elements: Defining Activity UI in the code

Marco Ronchetti
Università degli Studi di Trento

UI Programmatically



Hello, This is a view created programmatically!

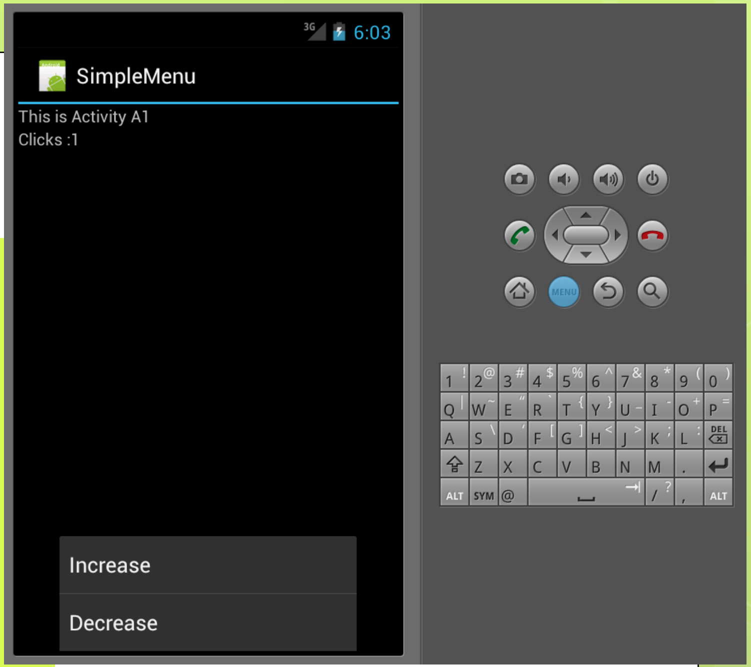
```
public class UIThroughCode extends Activity {
  LinearLayout lLayout;
  TextView tView;
  /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      lLayout = new LinearLayout(this);
      lLayout.setOrientation(LinearLayout.VERTICAL);
      lLayout.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
                              LayoutParams.MATCH_PARENT));
      tView = new TextView(this);
      tView.setText("Hello, This is a view created programmatically! ")");
      tView.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
                             LayoutParams.WRAP_CONTENT));
      lLayout.addView(tView);
      setContentView(lLayout);
   }
}
```

From http://saigeethamn.blogspot.it

Basic UI elements: Menus, a deeper insight

Marco Ronchetti
Università degli Studi di Trento

OptionMenu



```
public class A1 extends Activity {
                ...
    public boolean onCreateOptionsMenu(Menu menu){
        super.onCreateOptionsMenu(menu);
        int base=Menu.FIRST;
        MenuItem item1=menu.add(base,1,1,"Increase");
        MenuItem item2=menu.add(base,2,2,"Decrease");
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item) {
                ...
    }
```

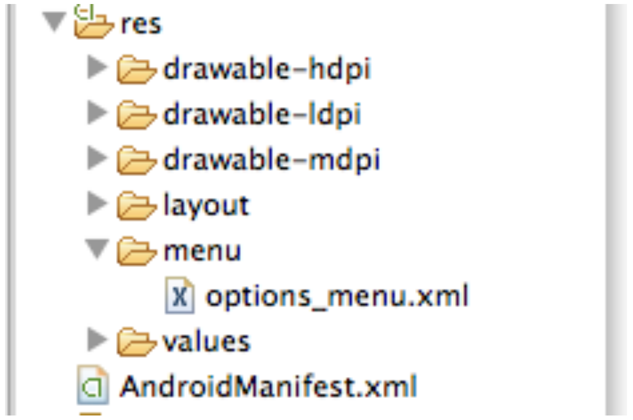
Menu is created
In code

```
public boolean onCreateOptionsMenu(Menu menu){
        super.onCreateOptionsMenu(menu);
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.option_menu, menu);
       return true;
    }
```

Menu is created
From XML

option_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
        <item android:id="@+id/increase"
                  android:title="@string/increase" />
        <item android:id="@+id/decrease"
           android:title="@string/decrease" />
</menu>
```



```
▼ res
    ▶ drawable-hdpi
    ▶ drawable-ldpi
    ▶ drawable-mdpi
    ▶ layout
    ▼ menu
        options_menu.xml
    ▶ values
  AndroidManifest.xml
```

Dynamically changing menu

Menu onPrepareOptionsMenu() (Activity class).

you get the Menu object as it currently exists, and you can modify it (add, remove, or disable items).

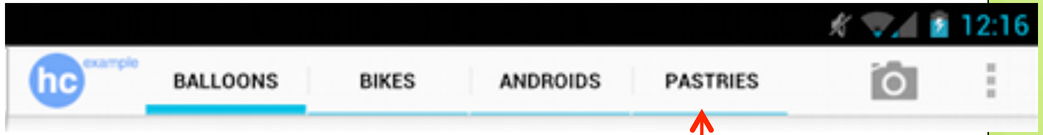
Android < 3.0:

- the system calls onPrepareOptionsMenu() each time the user opens the options menu.

Android >= 3.0

- When you want to perform a menu update, you must call invalidateOptionsMenu() to request that the system calls onPrepareOptionsMenu().

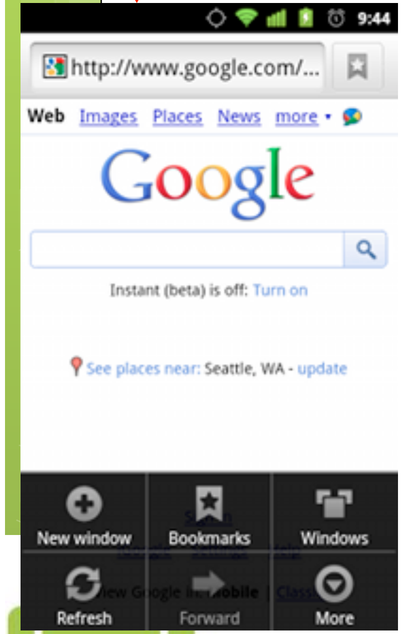
Menu types

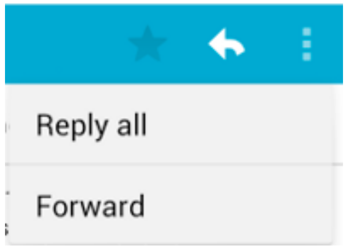


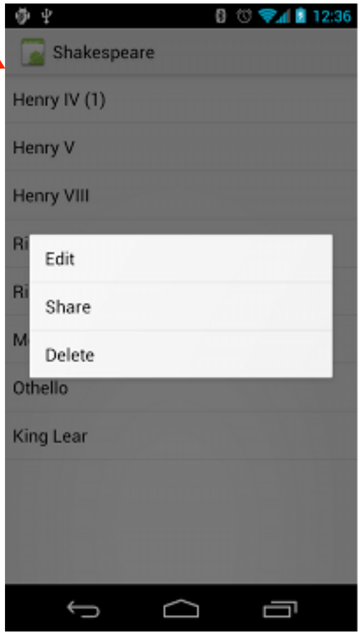
Traditional menus are awkward on a small screen.

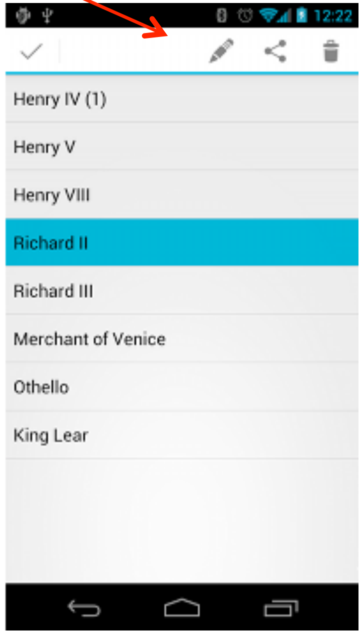
⇒ Three stages menus:

- Option Menu (< 3.0) / Action Bar (>=3.0)
- Context Menu (< 3.0) / Contextual Action mode(>=3.0)
- Popup Menu

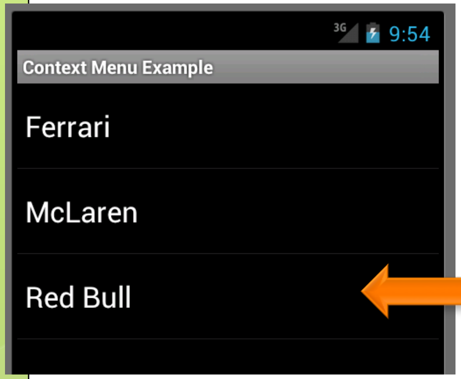

9:44
http://www.google.com/...
Web Images Places News more
Google
Instant (beta) is off: Turn on
See places near: Seattle, WA - update
New window
Bookmarks
Windows
Refresh
Forward
More



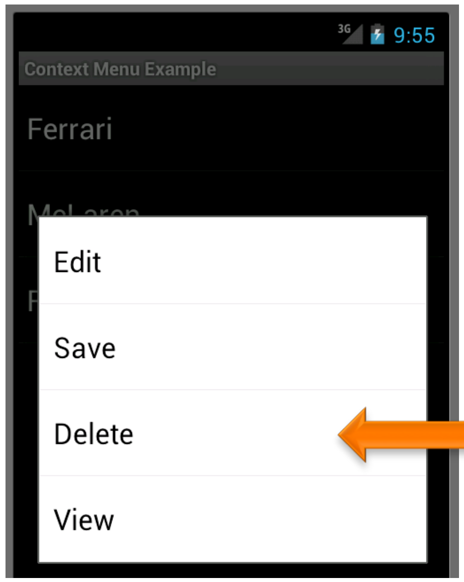




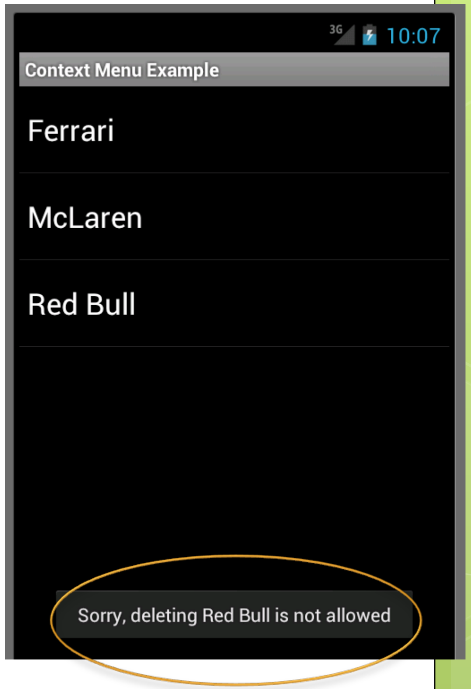
ContextMenu



LONG CLICK



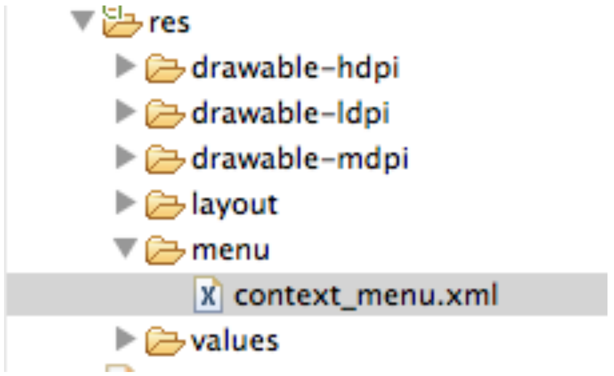
CLICK



Sorry, deleting Red Bull is not allowed

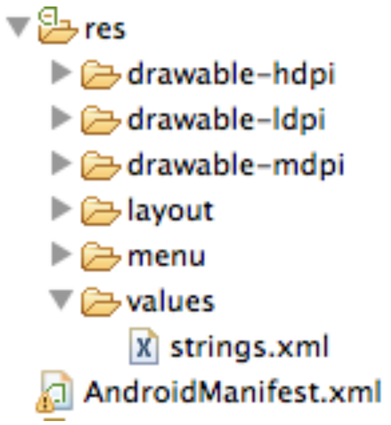
context_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
 xmlns:android="http://schemas.android.com/apk/res/android">
        <item android:id="@+id/edit"
                    android:title="@string/edit" />
        <item android:id="@+id/save"
           android:title="@string/save" />
        <item android:id="@+id/delete"
           android:title="@string/delete" />
        <item android:id="@+id/view"
           android:title="@string/view" />
</menu>
```



Strings

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">ShowContextMenu</string>
   <string name="app_name">Context Menu Example</string>
   <string name="edit">Edit</string>
   <string name="save">Save</string>
   <string name="delete">Delete</string>
   <string name="view">View</string>
<string-array name="names">
            <item>Ferrari</item>
            <item>McLaren</item>
            <item>Red Bull</item>
</string-array>
</resources>
```



ContextMenu

```
public class ShowContextMenu extends ListActivity {
@Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setListAdapter(new ArrayAdapter<String>(this,
             android.R.layout.simple_list_item_1,
             getResources().getStringArray(R.array.names)));
      registerForContextMenu(getListView());
   }

 public void onCreateContextMenu(ContextMenu menu,
                                             View v, ContextMenuInfo menuInfo) {
             super.onCreateContextMenu(menu, v, menuInfo);
             MenuInflater inflater = getMenuInflater();
             inflater.inflate(R.menu.context_menu, menu);
   }
```

Responding to event

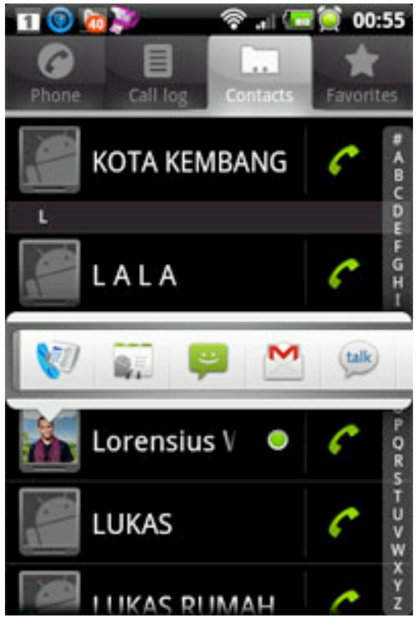
```
public boolean onContextItemSelected(MenuItem item) {
        AdapterContextMenuInfo info =
                (AdapterContextMenuInfo) item.getMenuInfo();
        String[] names = getResources().getStringArray(R.array.names);
        switch(item.getItemId()) {
        case R.id.delete:
          Toast.makeText(this,
                Toast t=Toast.makeText(this, "Sorry, deleting " +
                ((TextView)info.targetView).getText()
                + " is not allowed", Toast.LENGTH_LONG).show();
          return true;
        default:
          Toast.makeText(this, "You have chosen the " + item.getTitle() +
                " context menu option for " + names[(int)info.id],
                Toast.LENGTH_SHORT).show();
          return true;
        }
  }
```

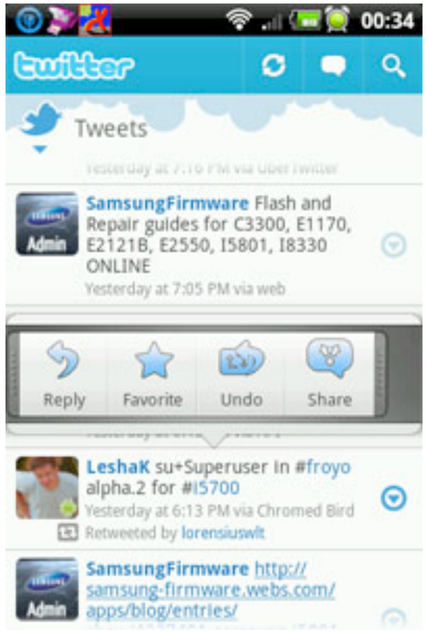
3.5 sec

2.0 sec

```
for (int i=0; i < 2; i++){ Toast.makeText(this, "blah", Toast.LENGTH_LONG).show(); }
```

A similar concept: QuickAction





SamsungFirmware Flash and Repair guides for C3300, E1170, E2121B, E2550, I5801, I8330 ONLINE
Yesterday at 7:05 PM via web

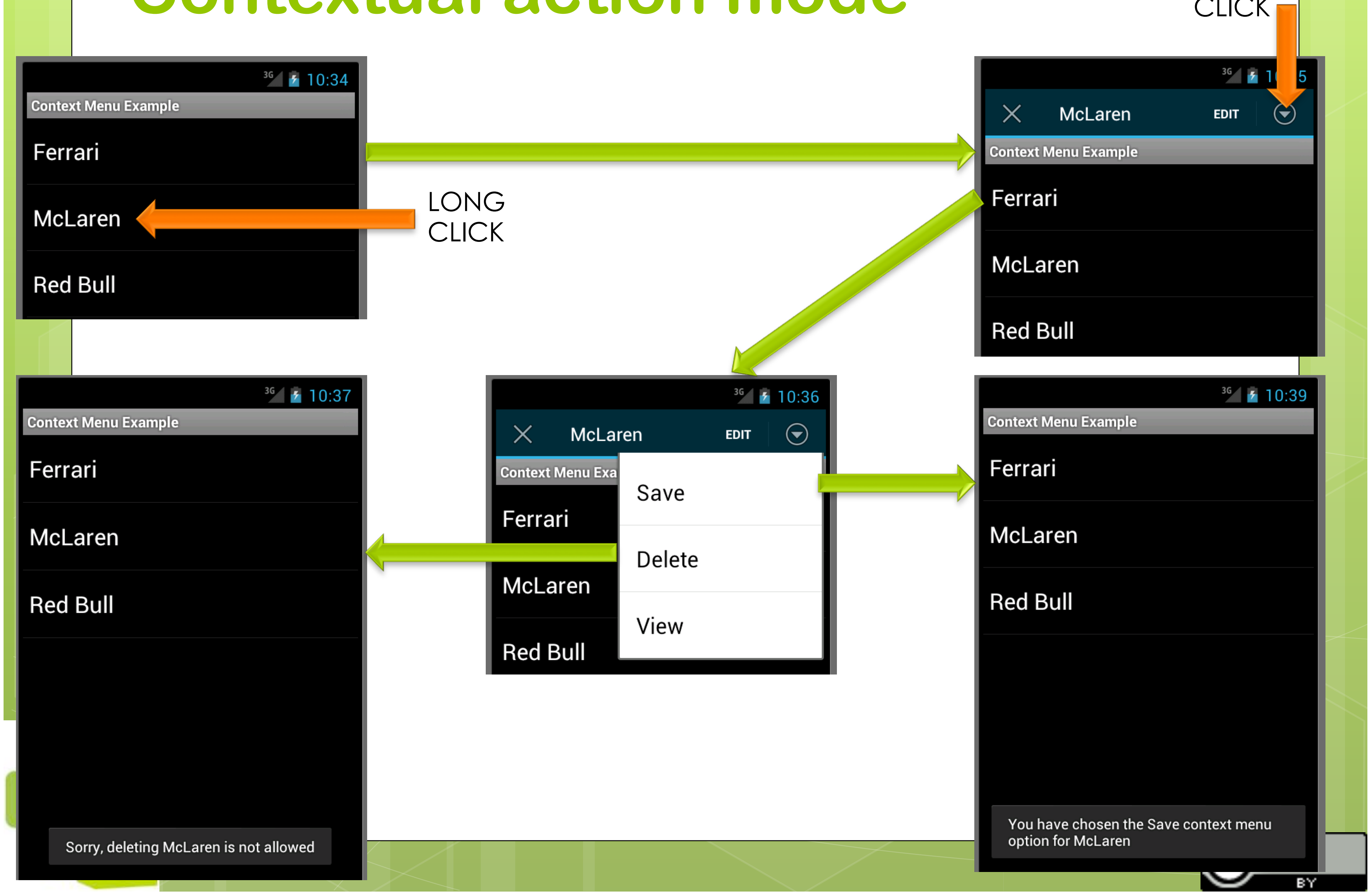
LeshaK su+Superuser in #froyo alpha.2 for #I5700
Yesterday at 6:13 PM via Chromed Bird
Retweeted by lorensiuswlt

SamsungFirmware http://samsung-firmware.webs.com/apps/blog/entries/

displays contextual actions in a list view

Not (yet) a standard API!
For a hint in the implementation see
http://www.londatiga.net/it/how-to-create-quickaction-dialog-in-android/

Contextual action mode



Sorry, deleting McLaren is not allowed

You have chosen the Save context menu option for McLaren

Contextual action mode

```
public class ShowContextMenu extends ListActivity {
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1,
                        getResources().getStringArray(R.array.names)));
      //registerForContextMenu(getListView());
      ListView listView=getListView();
      final Context ctx=this;
      listView.setChoiceMod(ListView.CHOICE_MODE_MULTIPLE_MODAL);
      listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {
            …
      });
  }
}
```

MultiChoiceModeListener

```
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {
     public boolean onCreateActionMode(ActionMode mode, Menu menu) {
              MenuInflater inflater = getMenuInflater();
              inflater.inflate(R.menu.context_menu, menu);
              mode.setTitle("selection");
              return true;
     }
     public void onDestroyActionMode(ActionMode mode) {
          // Here you can make any necessary updates to the activity when
          // the CAB is removed. By default, selected items are deselected/unchecked
    }
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
         // Here you can perform updates to the CAB due to an invalidate() request
         return true;
    }
```

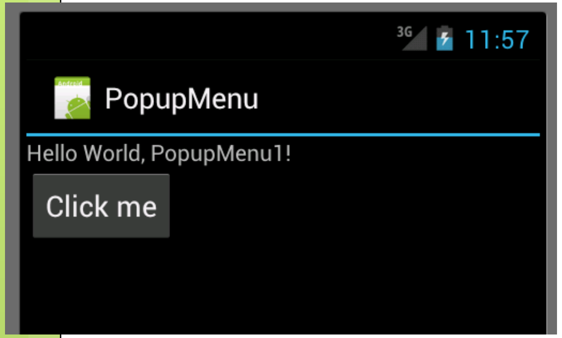
MultiChoiceModeListener

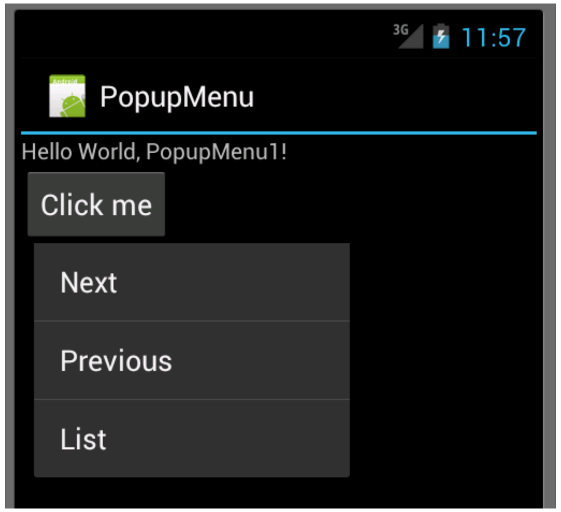
```
public void onItemCheckedStateChanged(ActionMode mode,
                        int position, long id, boolean checked) {
          // Here you can do something when items are selected/de-selected,
          // such as update the title in the CAB
          String[] names = getResources().getStringArray(R.array.names);
          if (checked) mode.setTitle(names[position]);
   }
```

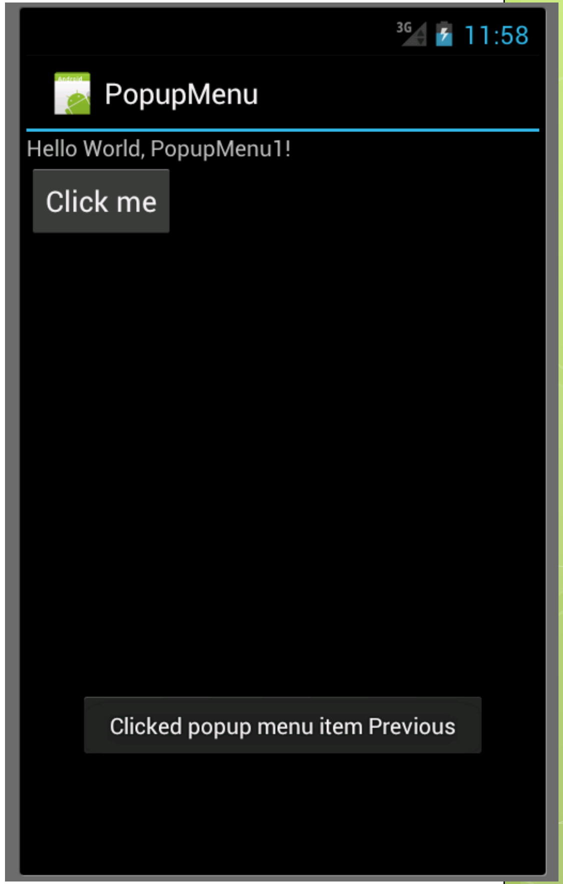
MultiChoiceModeListener

```
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch(item.getItemId()) {
            case R.id.delete:
                Toast t=Toast.makeText(ctx, "Sorry, deleting " +
                        mode.getTitle()  + " is not allowed”, Toast.LENGTH_LONG);
                t.show();
                mode.finish();
                return true;
            default:
                Toast.makeText(ctx, "You have chosen the " + item.getTitle() +
                            " context menu option for " + mode.getTitle(),
                             Toast.LENGTH_LONG).show();
                mode.finish();
                return true;
            }
}
```

PopupMenu







Clicked popup menu item Previous

Popup Menu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/
android">
          <item android:id="@+id/next"
                  android:title="@string/next" />
          <item android:id="@+id/previous"
             android:title="@string/previous" />
          <item android:id="@+id/list"
             android:title="@string/list" />
</menu>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
  "http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="showPopup"
    android:text="Click me" />
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello World, PopupMenu1!</string>
  <string name="app_name">PopupMenu</string>
  <string name="next">Next</string>
  <string name="previous">Previous</string>
  <string name="list">List</string>
</resources>
```

Popup Menu

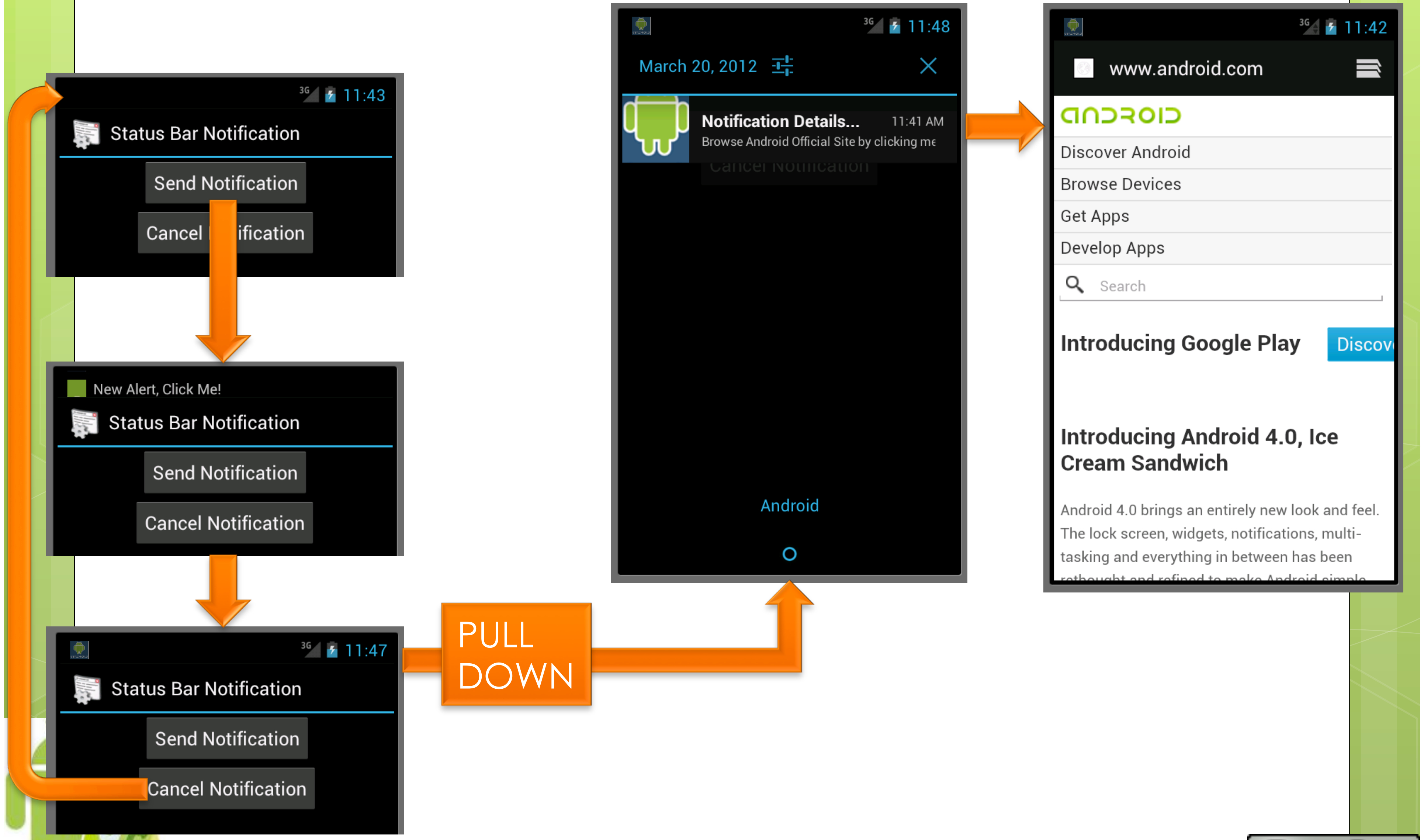
```
public class PopupMenu1 extends Activity {
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
```

```
public void showPopup(View button) {
      PopupMenu popup = new PopupMenu(this, button);
      popup.getMenuInflater().inflate(R.menu.popup, popup.getMenu());
      popup.setOnMenuItemClickListener(new
            PopupMenu.OnMenuItemClickListener() {
         public boolean onMenuItemClick(MenuItem item) {
            Toast.makeText(PopupMenu1.this, "Clicked popup menu item " +
                  item.getTitle(),  Toast.LENGTH_LONG).show();
            return true;
         }
      });
      popup.show();
   }
}
```

Notification

Marco Ronchetti
Università degli Studi di Trento

Notification Bar



SimpleNotification

```
public class SimpleNotification extends Activity {
  private NotificationManager nm;
  private int SIMPLE_NOTIFICATION_ID;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    nm = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    final Notification notifyDetails = new Notification(
          R.drawable.android,"New Alert, Click Me!",
          System.currentTimeMillis());
    Button cancel = (Button)findViewById(R.id.cancelButton);
    cancel.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
          nm.cancel(SIMPLE_NOTIFICATION_ID);
      }});}
```

Adapted from http://saigeethamn.blogspot.it

SimpleNotification – part 2

```
    Button start = (Button)findViewById(R.id.notifyButton);
    start.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        Context context = getApplicationContext();
        CharSequence contentTitle = "Notification Details...";
        CharSequence contentText = "Browse Android Site by clicking me";
        Intent notifyIntent = new Intent
           (android.content.Intent.ACTION_VIEW,
           Uri.parse("http://www.android.com"));
        PendingIntent intent =
        PendingIntent.getActivity(SimpleNotification.this, 0, notifyIntent,
            android.content.Intent.FLAG_ACTIVITY_NEW_TASK);
        notifyDetails.setLatestEventInfo(context, contentTitle,
            contentText, intent);
        nm.notify(SIMPLE_NOTIFICATION_ID, notifyDetails);
        }
    });
}}
```

Screen properties

Marco Ronchetti
Università degli Studi di Trento

Screen related terms and concepts

Resolution The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

Screen size Actual physical size, measured as the screen's diagonal.

Screen density The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch).

Orientation The orientation of the screen from the user's point of view. This is either landscape or portrait, meaning that the screen's aspect ratio is either wide or tall, respectively. Not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device.

Screen Sizes and Densities

Android divides the range of actual screen sizes and densities into:

A set of four generalized **sizes**:

xlarge at least 960dp x 720dp
large at least 640dp x 480dp
normal at least 470dp x 320dp
small at least 426dp x 320dp

A set of four generalized **densities**:

Low density (120 dpi), ldpi
Medium density (160 dpi), mdpi
High density (240 dpi), hdpi
Extra high density (320 dpi), xhdpi

Density-independent pixel (dp)

A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.

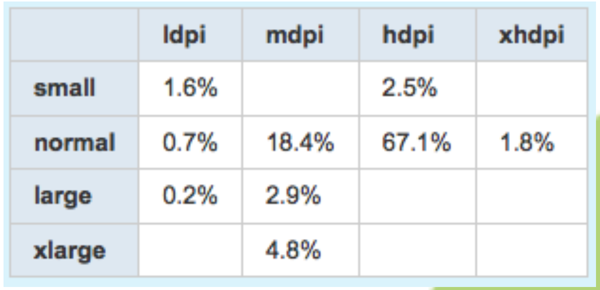
The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, (baseline for a "medium" density screen).

At runtime, the system transparently handles any scaling of the dp units based on the actual density of the screen in use.

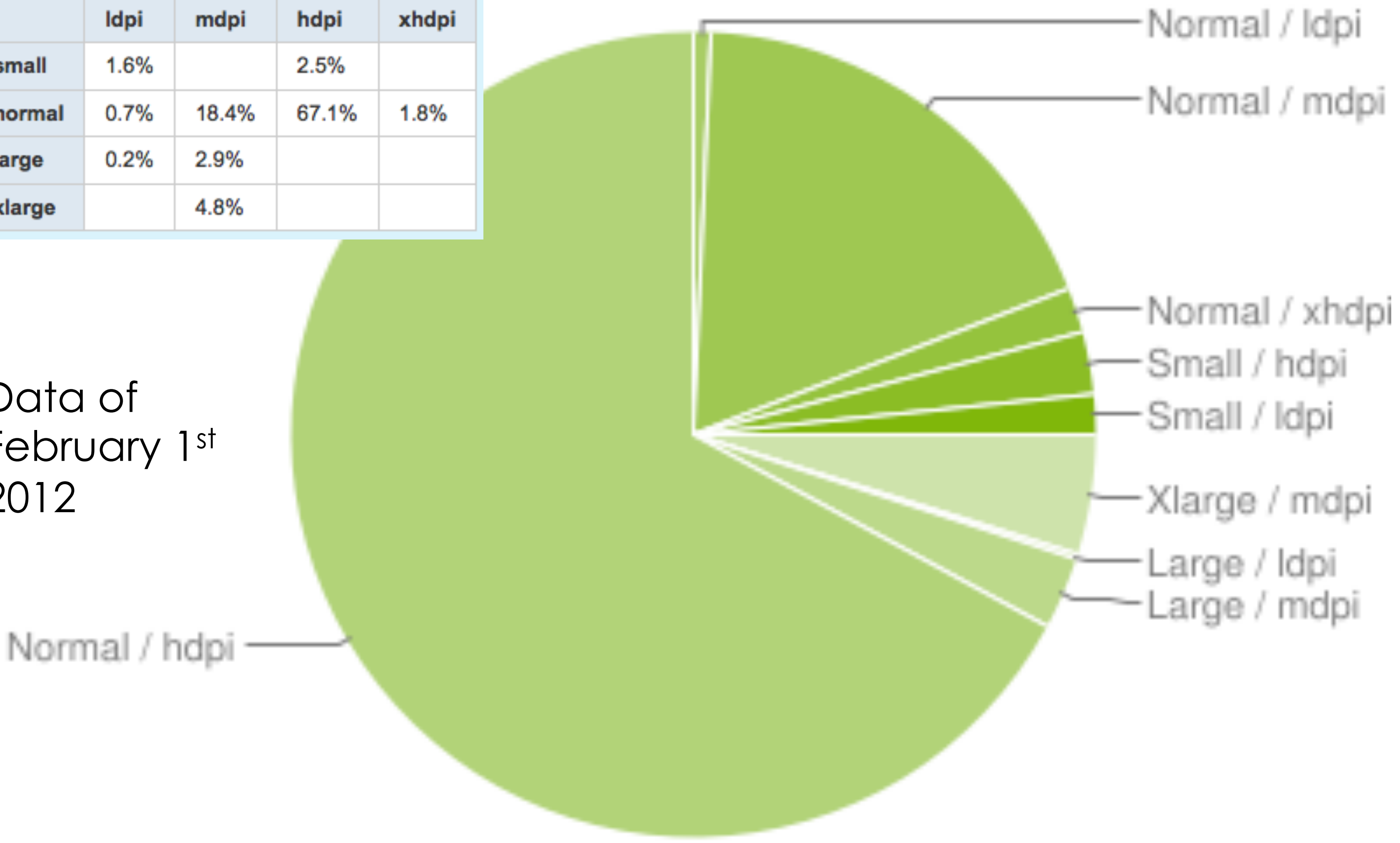
px = dp * (dpi / 160). E.g.: on a 240 dpi screen, 1 dp equals 1.5 physical pixels.

You should always use dp units when defining your application's UI, to ensure proper display on screens with different densities.

Screen Sizes and Densities



	ldpi	mdpi	hdpi	xhdpi
small	1.6%		2.5%	
normal	0.7%	18.4%	67.1%	1.8%
large	0.2%	2.9%		
xlarge		4.8%		



Data of February 1st 2012

http://developer.android.com/resources/dashboard/screens.html

Support of multiple versions

Marco Ronchetti
Università degli Studi di Trento

http://android-developers.blogspot.it/2010/07/how-to-have-your-cupcake-and-eat-it-too.html

http://android-developers.blogspot.it/2010/06/future-proofing-your-app.html

Fragments

Marco Ronchetti
Università degli Studi di Trento

Fragments

http://developer.android.com/guide/topics/fundamentals/fragments.html

Services

Marco Ronchetti
Università degli Studi di Trento

Service

An application component that can perform long-running operations in the background and does not provide a user interface.

Another application component can start a service and it will continue to run in the background even if the user switches to another application.

Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Caution: A service runs in the main thread of its hosting process—the service does not create its own thread and does not run in a separate process (unless you specify otherwise).
If your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should create a new thread within the service to do that work. By using a separate thread, you will reduce the risk of Application Not Responding (ANR) errors and the application's main thread can remain dedicated to user interaction with your activities.

Service

A service can essentially take two forms:

Started

A service is "started" when an application component (such as an activity) starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

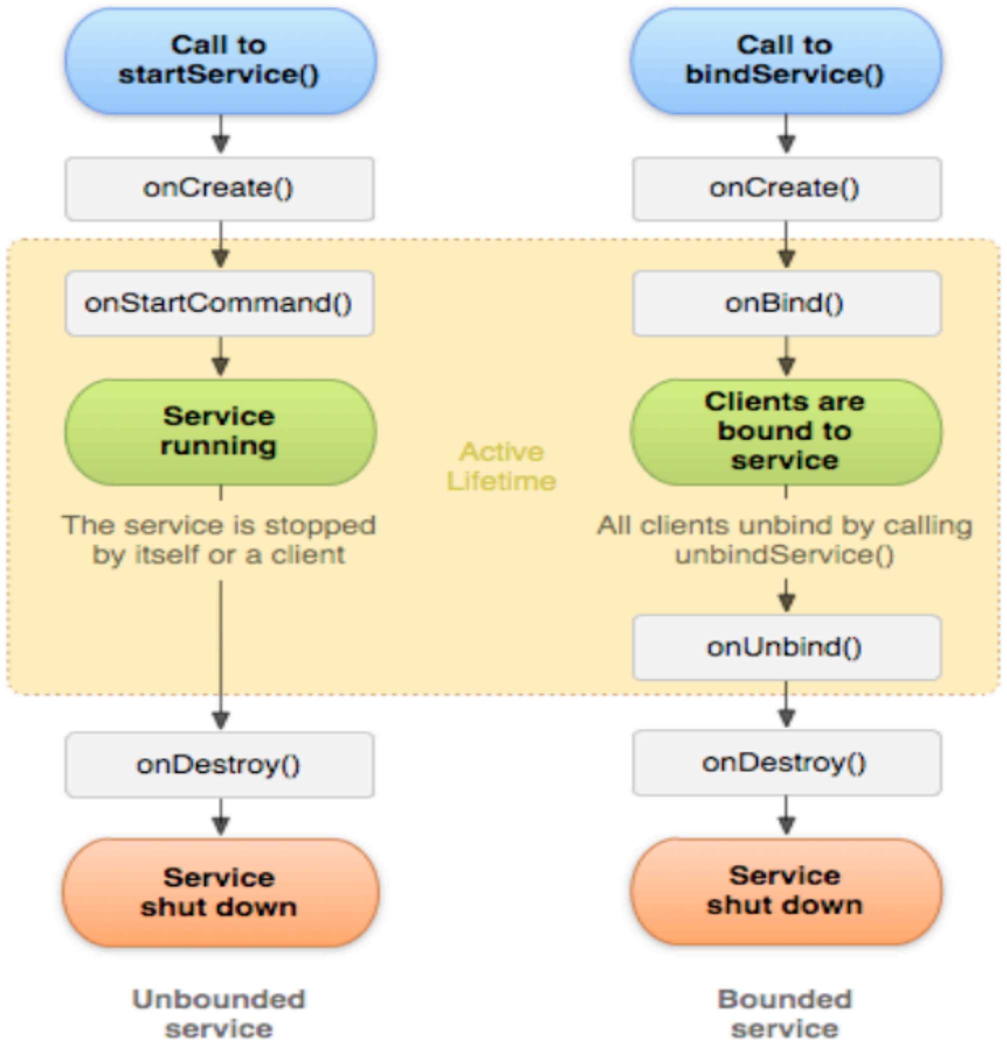
Bound

A service is "bound" when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Although this documentation generally discusses these two types of services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding. It's simply a matter of whether you implement a couple callback methods: onStartCommand() to allow components to start it and onBind() to allow binding.

Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application), in the same way that any component can use an activity—by starting it with an Intent. However, you can declare the service as private, in the manifest file, and block access from other applications.

Service lifecycle



```
    int mStartMode;       // indicates how to behave if the service is killed
    IBinder mBinder;      // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

public void onCreate()
        // The service is being created
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
public void onDestroy() {
        // The service is no longer used and is being destroyed
```

Adapters: a deeper insight

Marco Ronchetti
Università degli Studi di Trento

http://developer.android.com/resources/tutorials/views/index.html

http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/view/index.html

http://developer.android.com/training/improving-layouts/index.html

http://developer.android.com/guide/topics/ui/declaring-layout.html

Basic UI elements: Hello i18N

Marco Ronchetti
Università degli Studi di Trento

http://developer.android.com/resources/tutorials/localization/index.html

The Zygote

http://coltf.blogspot.com/p/android-os-processes-and-zygote.html http://www.slideshare.net/RanNachmany/manipulating-android-tasks-and-back-stack

http://www.vogella.de/articles/Android/article.html

http://www.vogella.de/articles/AndroidInternals/article.html

http://benno.id.au/blog/2007/11/13/android-under-the-hood

http://blog.vlad1.com/2009/11/19/android-hacking-part-1-of-probably-many/

http://www.slideshare.net/retomeier/being-epic-best-practices-for-building-android-apps

Fragment

A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running (it is in the resumed lifecycle state), you can manipulate each fragment independently, such as add or remove them. When you perform such a fragment transaction, you can also add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred. The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the Back button.

When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout. You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element, or from your application code by adding it to an existing ViewGroup. However, a fragment is not required to be a part of the activity layout; you may also use a fragment without its own UI as an invisible worker for the activity.

View

the basic building block for user interface components, similar to the Java AWT Component.

A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.)

Broadcast receiver

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object. For more information, see the BroadcastReceiver class.

Content Provider

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.

When you want to access data in a content provider, you use the ContentResolver object in your application's Context to communicate with the provider as a client. The ContentResolver object communicates with the provider object, an instance of a class that implements ContentProvider. The provider object receives data requests from clients, performs the requested action, and returns the results.

You don't need to develop your own provider if you don't intend to share your data with other applications. However, you do need your own provider to provide custom search suggestions in your own application. You also need your own provider if you want to copy and paste complex data or files from your application to other applications.

Android itself includes content providers that manage data such as audio, video, images, and personal contact information. You can see some of them listed in the reference documentation for the android.provider package.

Best practices

Marco Ronchetti
Università degli Studi di Trento

http://developer.android.com/guide/practices/design/performance.html

Screen properties

Marco Ronchetti
Università degli Studi di Trento

Android design

http://developer.android.com/design/index.html