Android Development

SmartPhone Application

4、User Interface

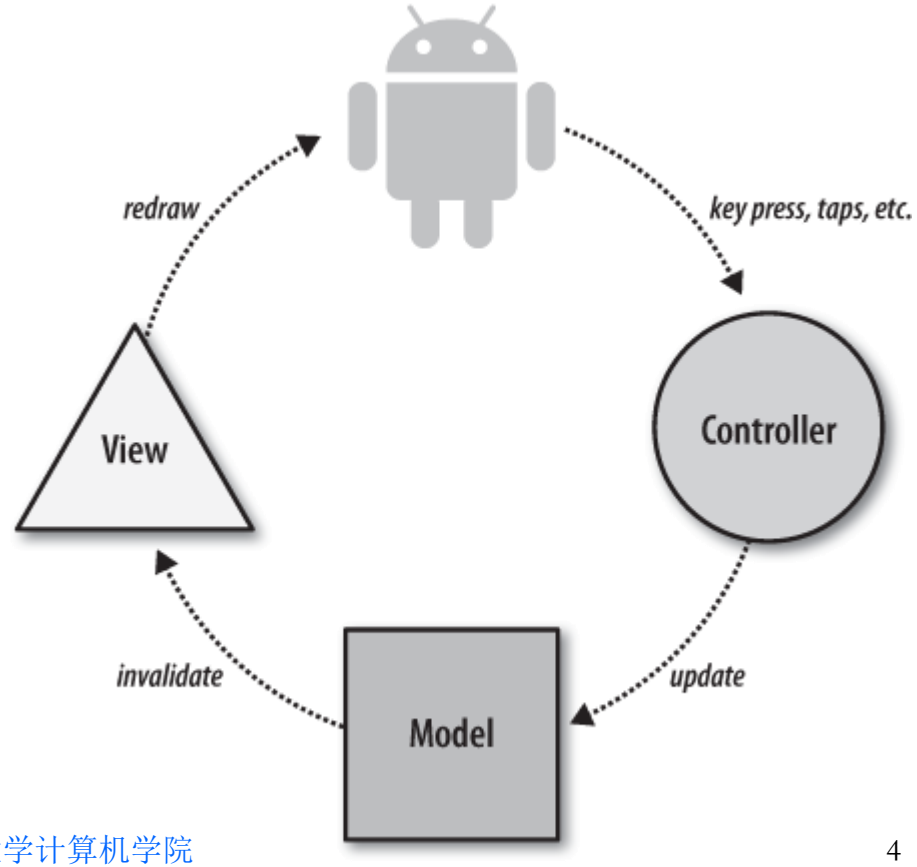
SmartPhone Application

Android GUI Architecture

- The Android environment adds yet another Graphical User Interface (GUI) toolkit to the Java ecosphere, joining AWT, Swing, SWT, and J2ME (leaving aside the web UI toolkits).
- The GUI is single-threaded, event-driven, and built on a library of nestable components.

MVC

- The Android UI framework is organized around the common Model-View-Controller pattern.



Screen

- HVGA-L：480x320
- HVGA-P：320x480
- QVGA-L：320x240
- QVGA-P：240x320
- WVGA-L：800x480

widget

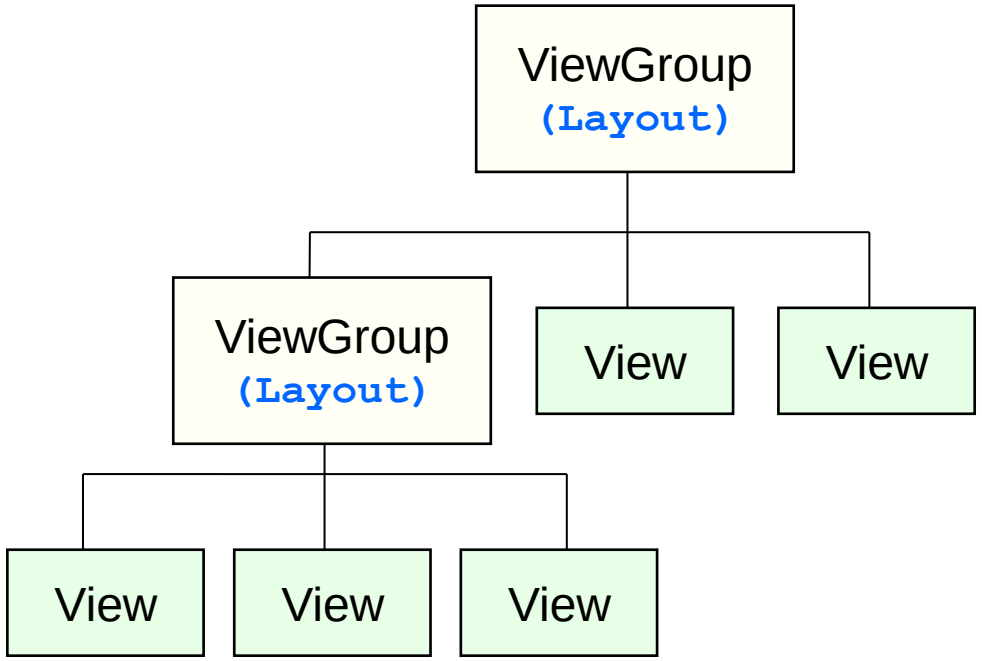
- subclasses of android.view.View.

User Interface Guidelines

- Activity and Task Design
  - Activities are the basic, independent building blocks of applications. As you design your application's UI and feature set, you are free to re-use activities from other applications as if they were yours, to enrich and extend your application.

Android UI: layout

- A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers.
- ViewGroup arranges their children by Layouts.



View

- View: An object that knows how to draw itself to the screen.
- ViewGroups: Views that can contain or group other Views.
- Layout: Views are arranged and displayed on the screen according to.
  - Views and Layouts both have attributes that can either be defined in Java source code or in the XML file associated with the Activity that uses the View or Layout.
  - When the attributes are in an XML file, they are "inflated" at runtime, meaning that they are applied to their respective Views by the Android framework to determine how the Views look and operate.

TextView and EditText

- A TextView: a place to display a text string. The vanilla TextView is for display only, whereas EditText is a predefined subclass of TextView that includes rich editing capabilities.
- Each TextView has the attributes: height, width, font, text color, background color, and so forth. TextViews also have some useful unique attributes:
  - autoLink: If set (true), finds URLs in the displayed text and automatically converts them to clickable links.
  - autoText: If set (true), finds and corrects simple spelling errors in the text.
  - editable: If set (true), indicates that the program has defined an input method to receive input text (default is false for TextView, and true for EditText).

Button and ImageButton

- The Button View is just a button, printed with some text to identify it, that the user can click to invoke some action. an OnClickListener method will be executed when the Button click.
- the ImageButton View: to use a button with an image on it rather than one with text.

Button

- Main.xml

```
<Button android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/btn_name"
        />
```

- Click callback

```
show= (TextView)findViewById(R.id.TextView01);
press=(Button)findViewById(R.id.Button01);

press.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
                show.setText("Hi , Google Android!");
        }
});
```

CheckBoxes, RadioButtons, and Spinners

- The purpose of The Views is to allow the user to choose from multiple options.
  - CheckBoxes are typically used when you want to offer multiple selections with a yes/no or true/false choice for each.
  - RadioButtons are used when only one choice is allowed at a time.
  - Spinners are similar to combo boxes in some frameworks. A combo box typically displays the currently selected option, along with a pull-down list from which the user can click on another option to select it.
- Android has adapted these familiar components to make them more useful in a touchscreen environment.

ViewGroups

- ViewGroups are Views that contain child Views. Each ViewGroup class embodies a different set of assumptions about how to display its child Views. All ViewGroups descend from the android.view.ViewGroup class.
- Layouts are a subset of ViewGroups.

Gallery and GridView

- The Gallery ViewGroup displays multiple items in a horizontally scrolling list.
  - The currently selected item is locked in the center of the screen.
  - Any items that approach the edge of the screen begin to fade, giving the user the impression that there may be more items "around the corner.“
  - The user can scroll horizontally through the items within the gallery.
- This ViewGroup is useful when you want to present a large set of possible choices to the user without using too much screen real estate.

ListView and ListActivity

- ListView is similar to Gallery, but uses a **vertically** scrolling list in place of Gallery's horizontally scrolling list. To create a ListView that takes up the entire screen, Android provides the ListActivity class.

ScrollView

- A ScrollView is a container for another View that lets the user scroll that View vertically (a scrollbar is optional).
- A ScrollView often contains a LinearLayout, which in turn contains the Views that make up the form.
- Don't confuse ScrollView with ListView. Both Views present the user with a scrollable set of Views, but
  - The ListView is designed to display a set of similar things, such as the cheeses.
  - The ScrollView allows an arbitrary View to scroll vertically.
    - The Android documentation warns that one should never house a ListView within a ScrollView, because that defeats the performance optimizations of a ListView.
- A ScrollView is a FrameLayout, which means that it can have only one child View. The most popular View for this purpose is a LinearLayout.

TabHost

- Most modern UIs provide an interface element that lets the user flip through many pages of information quickly using tabs, with each "screen" of information available when its tab is pressed. Android's option is the TabHost View.

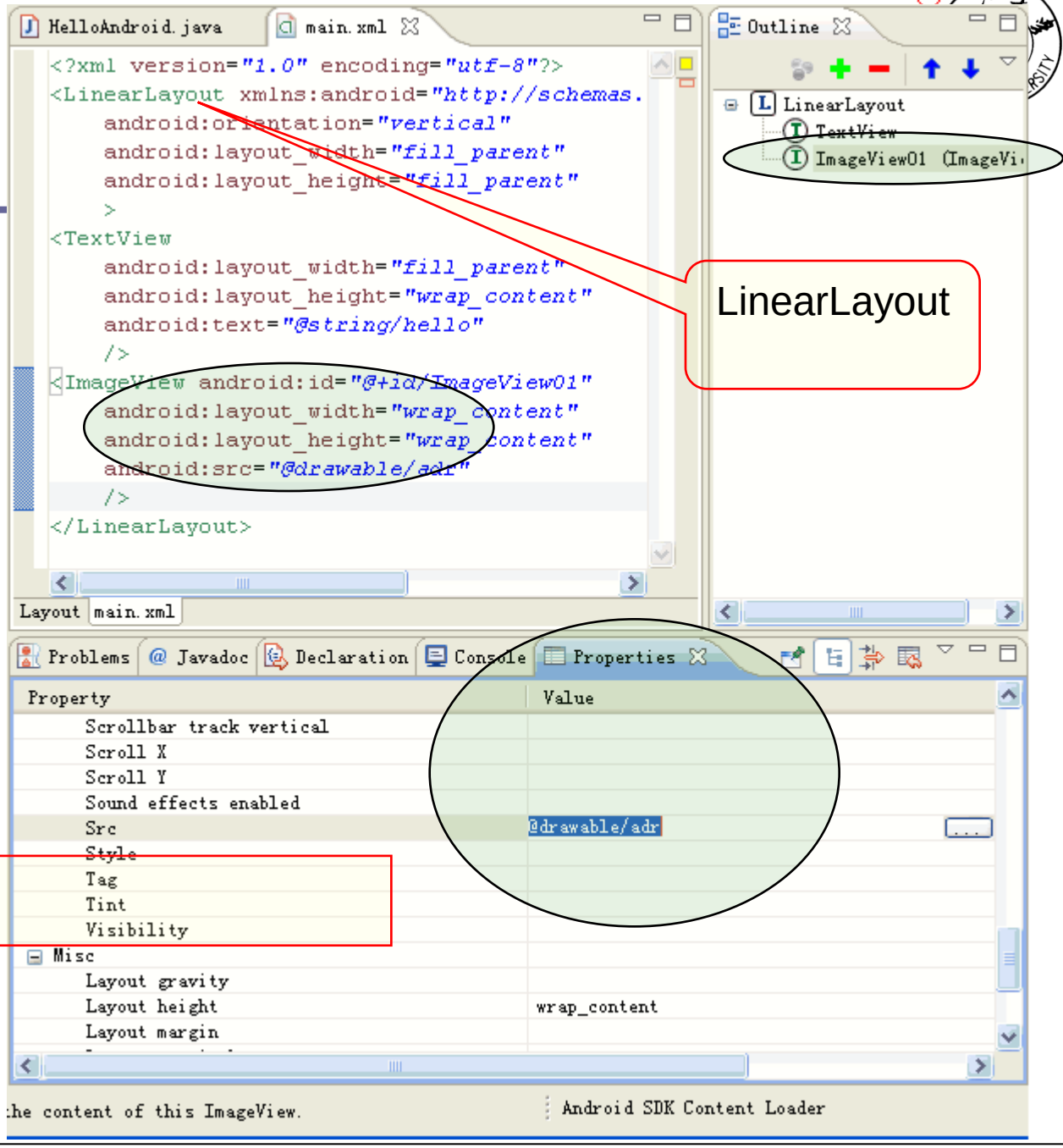
Layout

- AbsoluteLayout
- FrameLayout
- GridView
- LinearLayout
- ListLayout
- RadioGroup
- TableLayout
- ……

- Layouts are Android's solution to the variety of screens that come on Android devices: they can have
  - different pixel densities
  - different dimensions
  - different aspect ratios
- Typical Android devices even allow changing the screen orientation (portrait or landscape) while applications are running, so the layout infrastructure needs to be able to respond on the fly.
- Layouts are intended to give developers a way to express the physical relationship of Views as they are drawn on the screen.
- As Android inflates the Layout, it uses the developer requests to come up with a screen layout that best approximates what the developer has asked for.

Image display

- Add ImageView
- Arrange the position by layout.



LinearLayout

- LinearLayouts are used extensively in Android applications.
- A LinearLayout asks that the contained Views be layed out as either a series of rows (vertical LinearLayout) or a series of columns (horizontal LinearLayout).
  - In a vertical LinearLayout, all the rows are the same width (the width of the widest child).
  - In a horizontal LinearLayout, there is one row of Views, all the same height (the height of the tallest child).

LinearLayout

- LinearLayout
  - android:orientation="vertical“
  - android:orientation="horizontal“
- In addition to the usual dimensional parameters for child Views (width, height, padding).

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
```

FrameLayout

- The FrameLayout is sort of a null layout specification. It reserves space on the screen for a single View to be drawn, and the View is always located at the upper left of the space.
- There is no way to specify a different location for the View, and there can be only one View in the Layout.
- If more than one View is defined in the layout file, they are just drawn on top of each other, all pinned to the upper-left corner.

TableLayout

- A TableLayout is just what you'd expect: it lays out the included Views in the form of a table (similar to an HTML table).
- We can create a table of TextViews to show how you would create that kind of screen for an application.

```
<TableLayout …… >
      <TableRow  ……>
... ...
      </TableRow>
      <TableRow  ……>
... ...
      </TableRow>
</TableLayout>
```

AbsoluteLayout

- An AbsoluteLayout puts views on the screen wherever you tell it to.
  - It doesn't try to resize anything, and it doesn't try to line anything up; it just puts things where it's told.
- AbsoluteLayout bypasses most of the layout manager, and while your application may look perfect on the device you used for development, the odds are very good that it will look terrible on other Android devices.
- As with any dimension in a layout file, the positions can be expressed in:
  - pixels (px)
  - device-independent pixels (dp)
  - scaled pixels (sp)
  - inches (in)
  - millimeters (mm)
  - dimension has to be a floating-point number.

RelativeLayout

- The advantage of RelativeLayout is that you can express the relative positioning of the Views in the screen, and the layout manager will do its best to fit them all on the screen in the proper relations.

Colors in Android

- Colors in Android are defined by a pound sign (#) followed by four 8-bit integers in hexadecimal:
- Alpha
  - The transparency of the resulting color, FF being completely opaque and 0 being completely transparent.
- Red
  - Red's contribution to the resulting color, FF being fully on and 0 meaning no red.
- Green
  - Green's contribution to the resulting color, FF being fully on and 0 meaning no green.
- Blue
  - Blue's contribution to the resulting color, FF being fully on and 0 meaning no blue.
- Common colors are also defined as global constants for use in Java.

Color constant

- android.graphics.Color
  - Color.BLACK
  - Color.BLUE
  - Color.CYAN
  - Color.DKGRAY
  - Color.GRAY
  - Color.GREEN
  - Color.LTGRAY
  - Color.MAGENTA
  - Color.RED
  - Color.TRANSPARENT
  - Color.WHITE
  - Color.YELLOW

Font Typeface

- android.graphics.Typeface
  - Typeface.create(Typeface family, int style)
  - Typeface.setTypeface (Typeface tf, int style)
  - Typeface.setTypeface(Typeface tf)
- Typeface(4)
  - BOLD、BOLD_ITALIC、ITALIC、NORMAL
- Style(5)
  - DEFAULT、DEFAULT_BOLD、MONOSPACE、SANS_SERIF、SERIF

Dimensions in Android

- Android actually offers a rich set of dimensions to choose from:
- px (pixels)
  - If a dimension is set at 10px, it will be exactly 10 pixels long, no matter what the physical size (and physical density) of the pixels on the display.
- dip or dp (device-independent pixels)
  - In an effort to make it easier to adapt applications to different pixel densities, dimensions can be expressed in device-independent pixels (sometimes also called "density-independent pixels").
- sp (scaled pixels)
  - Scaled pixels are a lot like dp's, but they are intended for elements that need finer control over the density scaling factor, such as text.
- pts (points)
  - This is used to express text size in points, just as you would in any text editor. Points are a fixed dimension (roughly 1/72nd of an inch), so the text will appear the same size on any display.
- in (inches)
  - The dimension in inches.
- mm (millimeters)
  - This is also metric this time.

Create a callback

- When a user clicks a button, Android sends an event to its OnClickListener listener. In this code, we set the Button's behavior by setting its OnClickListener to the method that we immediately define, onClick.

User input：setting listener

- View.OnClickListener
  - OnClick()
- View.OnLongClickListener
  - OnLongClick()
- View.OnFocusChangeListener
  - OnFocusChange()
- View.OnKeyListener
  - OnKey()
- View.OnTouchListener
  - OnTouch()
- View.OnCreateMenuListener
  - OnCreateContextMenu()
- Setting a listener for a View to deal with its events

Event listener

- Button：

```
Button btn =(Button)findViewById(R.id.Button01);

... ... ...

btn.setOnClickListener(new OnClickListener(){
            public void onClick(View v) {
                  ……
            }
        });
```

Android menu

- Android has three menu:
  - options menu: main menu. Push "*home*" button to show it.
  - context menu: right-key, or popup menu. By push on it more than 2 seconds.
  - sub menu: sub menu, can be added to other two menus.
- android also provides the function of grouping menu items, can be similar to the function menu item into the same group. So we can set the group attributes by call setGroupCheckable, setGroupEnabled, setGroupVisible.

OptionMenu 1

- OptionMenu(main menu)
  - options menu can only display up to six menu options at the bottom of the screen. It is a Icon menu.
    - icon menu means it can not contain any checkable option.
  - More than 6 menu items will be bring up by “more icon menu”, it is called expanded menu。
  - options menu is built by call onCreateOptionsMenu.
  - onCreateOptionsMenu only be called to create a new menu.
    - We can change the options menu by setting onPrepareOptionsMenu function. This function will be called before the menu shows up.
  - onOptionsItemSelected will deal with the selected item in the menu.

OptionMenu 2

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
        <item  android:id="@+id/save"
               android:title="@string/save" />
        <item  android:id="@+id/cancel"
               android:title="@string/cancel" />
</menu>
```

ContextMenu

- A ContextMenu appears in response to a long click in a widget that supports it.
- The code required to add a contextual menu to an application is entirely analogous to that for the options menu shown earlier except that the respective methods are onCreateContextMenu and onContextItemSelected.
- In order to support contextual menus, a widget must be assigned a View.OnCreateContextMenuListener by calling its View method, setOnCreateContextMenuListener.
- Activity implements the View.OnCreateContextMenuListener interface.

Context Menu

- Context Menu(popup menu)
  - To show a context menu on long click
  - Context Menu – displays information related to a particular view on an activity. registerForContextMenu is used to register a context menu for a view in an activity.
  - context menu will call onCreateContextMenu to create the menu before display.
  - onContextItemSelected will deal with the selected item in the menu

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
      <item  android:id="@+id/save"
             android:title="@string/save"      />
      <item  android:id="@+id/cancel"
             android:title="@string/cancel"/>
</menu>
```

Submenu

- Submenu
  - A floating menu list.
  - be added to other two menus.
  - Does not support nested sub-menu

Add Menu

- Menu

```
public boolean onCreateOptionsMenu(Menu menu) {
// TODO Auto-generated method stub
boolean result = super.onCreateOptionsMenu(menu);
menu.add(0, INSERT_ID_Play, 0, R.string.menu_toPlay);
menu.add(0, INSERT_ID_Stop, 0, R.string.menu_toStop);
return result;
}//创建菜单
```

Thank you!