These slides are by Dr. Jaerock Kwon at Kettering University.

See http://mirlab.wordpress.com/people/jaerock-kwon/ for details.

Lecture 3
Creating User Interfaces

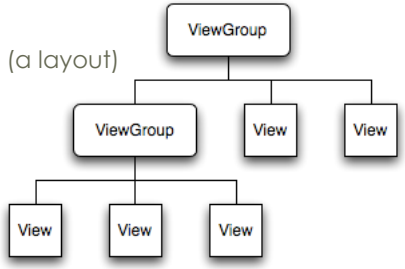
Today's Topics

- Views
- Layout
- UI Event
  - Event Listener
- Menu
- Dialog

Layout

View, ViewGroup, and Activity

- The user interface is built using View and ViewGroup objects.
- View Hierarchy
  - View object
    - The basic unit of user interface expression.
  - ViewGroup object
    - The base for subclasses called "layouts."
- Activity
  - To display a user interface, assign a View (a layout) to an Activity.
  - setContentView()
    - Activity must call this method.



Creating Activity UI w/ Views

- The `setContentView` method accepts either a layout resource ID or a single View instance.
- Example
  - 1) Using a layout resource
    - ```
      @override
      public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        **setContentView(R.layout.main);**
        TextView myTextView = (TextView)findViewById(R.id.myTextView);
      }
      ```
  - 2) Creating a UI layout in code
    - ```
      @override
      public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView myTextView = new TextView(this);
        myTextView.setText("Hello, Kettering");
        **setContentView(myTextView);**
      }
      ```


Layout

- The name of an XML element is respective to the Java class.
  - <TextView> element creates a TextView in your UI.
  - <LinearLayout> element creates a LinearLayout view group.
- Layout manager is the extension of the ViewGroup class
  - Used to position child controls for your UI.
- Layout classes
  - RelativeLayout
  - LinearLayout
  - FrameLayout
  - TableLayout
  - Gallery
    - Displays a single row of items in a horizontally scrolling list.

Common Layout Objects

- RelativeLayout
  - It lets child views specify their position relative to the parent view.
- FrameLayout
  - The simplest type of layout object.
  - It is a blank space on you screen that you can later fill with a single object.
- LinearyLayout
  - It aligns all children in a single direction – vertically or horizontally.
  - All children are stacked one after the other.
- TableLayout
  - It positions its children into rows and columns.
  - TableRow is the child view of a TableLayout

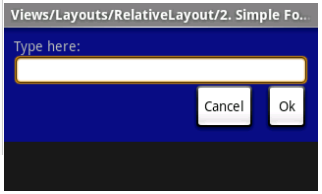
Element Size

wrap_content and fill_parent

- `wrap_content`
  - Sets the size of a View to the minimum required to contain the contents it displays.
- `match_parent`
  - Expands the View to fill the available space within the parent View.
- The width and height can be set by `wrap_content` and `match_parent` rather than an exact height or width in pixels.

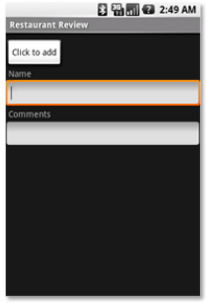
RelativeLayout Example

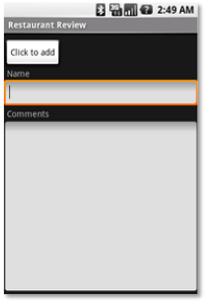

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:background="@drawable/blue"
                android:padding="10px" >

    <TextView android:id="@+id/label"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Type here:" />

    <EditText android:id="@+id/entry"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:background="@android:drawable/editbox_background"
              android:layout_below="@id/label" />

    <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
            android:text="OK" />

    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/ok"
            android:layout_alignTop="@id/ok"
            android:text="Cancel" />
</RelativeLayout>
```



LinearLayout Example

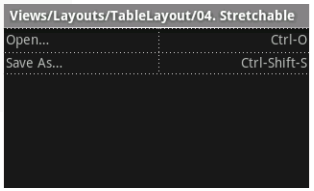
- Size of elements
  - match_parent vs. wrap_content
  - Text boxes have their widths: match_parent
  - Other elements have their width: wrap_content.
- Gravity (alignment):
  - The gravities of all elements are left.
- Weight:
  - The left version:
    0 for all UI components.
  - The right version:
    - Comments text box has 1.
  - If the Name text box has also 1, then
    - Two text boxes would have the same height.





TableLayout Example

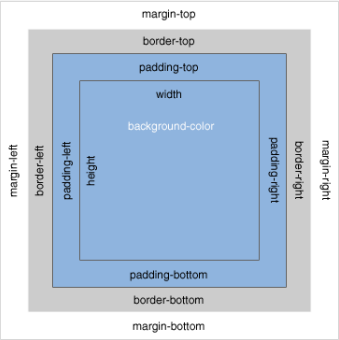

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="@string/table_layout_4_open"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_open_shortcut"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:text="@string/table_layout_4_save"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_save_shortcut"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```



Layout Parameters

Box Model for View Dimension

- **ViewGroups** provide **Margin**.
  - ViewGroup.LayoutParams.leftMargin
  - ViewGroup.LayoutParams.topMargin
  - ViewGroup.LayoutParams.rightMargin
  - ViewGroup.LayoutParams.bottomMargin
- **Views** support **Padding**.
  - GetPaddingLeft();
  - GetPaddingTop();
  - GetPaddingRight();
  - GetPaddingBotton();



Event Listener

UI Events

- Now you have some Views on the screen.
- How to connect user's interactions with Views?
- You need to do one of these
  - Event Listener
    - Define an event listener and register it to the View.
    - The View class contains a collection of nested listener interfaces.
    - Examples
      - View.On**Click**Listener
      - View.On**Touch**Listener
  - Override an existing callback method for the View.
    - When you've implemented your own UI class.

Event Listener

- For example,
  - A View is touched.
  - onTouchEvent() method is called on that object.
  - In order to intercept this event, we must extend the class and override the method.
  - This approach is OK for a single view.
  - What if there are many different types of views on one Activity?
  - Extending each class and override the method for each View are not practical.
- Event Listener!
  - An interface in the View class that contains a single callback method.
  - Set your own event handler to the Listener. Then Android framework will call it when a corresponding event occurs.

Event Listener

Callback methods and EventListener interface

- Callback method and Event listener interfaces
  - `onClick` `View.OnClickListener`
  - `onLongClick` `View.OnLongClickListener`
  - `onFocusChange` `View.OnFocusChangeListener`
  - `onTouch` `View.OnTouchListener`
  - `onCreateContextMenu` `View.OnCreateContextMenu`
- Examples

```
private OnClickListener myButtonListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    Button button = (Button)findViewById(R.id.myButton);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(myButtonListener);
    ...
}
```


Menu

Menu

Three types of application menus

- Options Menu
  - The primary menu for an Activity
  - When the user presses the device MENU key.
  - Two groups of Option Menu
    - Icon Menu
      - The menu items visible at the bottom of the screen.
      - Maximum of six menu items.
      - Icon menu items do not support checkboxes and radio buttons.
    - Expanded Menu
      - The vertical list of menu items exposed by the “More” menu item in the Icon Menu.
- Context Menu
  - A floating list of menu items.
- Submenu
  - A floating list of menu items that the user opens by pressing a menu item in the Options Menu or Context Menu.

Options Menu

Definition of Options Menu

- Define a menu and its items in an XML.
  - Create an XML file inside `res/menu/` directory.
- XML items
  - android:id
    - Unique id to the item.
  - android:title
    - Visible to the user.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item01" android:title="@string/item01"></item>
    <item android:id="@+id/item02" android:title="@string/item02"></item>
    <item android:id="@+id/item03" android:title="@string/item03"></item>
</menu>
```

Options Menu

Inflating a Menu Resource

- Use `MenuInflater.inflate()` to inflate a menu resource in `onCreateOptionsMenu()` callback method.

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


Options Menu

Selecting an Item

- When the user selects a menu item from Options Menu, `onOptionsItemSelected()` method will be called with `MenuItem` by the system.

```
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.item01:
        doSomething01();
        return true;
    case R.id.item02:
        doSomething02();
        return true;
    case R.id.item03:
        doSomething03();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```


Context Menu

- A context menu is displayed when the user long-presses an item.
- Conceptually similar to Right-click menu on a PC.
- Use `onCreateContextMenu()`.

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


Creating Submenus

- No nested submenus
  - A submenu cannot have another submenu.
- Adding a `<menu>` element as the child of an `<item>`.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item01" android:title="@string/item01">
        <!-- "item01" submenu -->
        <menu>
            <item android:id="@+id/item01_new"
                  android:title="@string/item01_new" />
            <item android:id="@+id/item02_open"
                  android:title="@string/item01_open" />
        </menu>
    </item>
    <item android:id="@+id/item02" android:title="@string/item02"></item>
    <item android:id="@+id/item03" android:title="@string/item03"></item>
</menu>
```

Dialog

Dialog

- A small window that appears in front of the current Activity.
- Four Dialog object types.
  - AlertDialog
    - A dialog that has buttons or selectable items.
  - ProgressDialog
    - A dialog that displays wheel or progress bar.
  - DatePickerDialog
    - A dialog that allows the user to select a date.
  - TimePickerDialog
    - A dialog that allows the user to select a time.

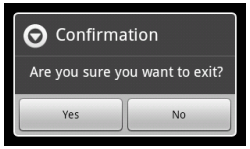
AlertDialog

- Use `onCreateDialog(int)` callback method to create dialogs.
- Use `showDialog(int)` to show a dialog.
- Call `onPrepareDialog(int, Dialog)` if you want to change any properties of the dialog.
- Example:

```
// 1. define an integer ID for your dialog
static final int DIALOG_ID = 0;

// 2. Then, define the onCreateDialog(int) callback
protected Dialog onCreateDialog(int id) {
    Dialog dialog;
    switch(id) {
    case DIALOG_ID :
        // Build a dialog!!
        break;
    default:
        dialog = null;
    }
    return dialog;
}

// 3. Now, you can show the dialog
showDialog(DIALOG_ID);
```


Creating an AlertDialog

- Use AlertDialog.Builder to make an AlertDialog.
  - A title
  - A text message
  - Button(s)
  - A list of selectable items.




```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setTitle("Confirmation")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                HelloKettering.this.finish();
           }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
           }
       });
AlertDialog alert = builder.create();
alert.show();
```

Supporting Multiple Screens

Variety of Android Devices

- Android runs on a variety of devices that offer different screen size and densities.
- Developers should make the effort to optimize your app for different screen size and densities.

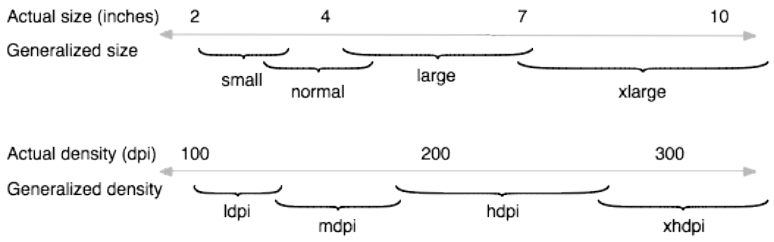
Screen

- Screen size
  - Actual physical size measured as the screen's diagonal.
  - All actual screen sizes into four generalized sizes:
    - small, normal, large, and extra large.
- Screen density
  - The quantity of pixels with a physical area of the screen. (usually referred to as dots per inch (dpi).
  - All actual screen densities into four generalized densities:
    - low, medium, high, and extra high.
- 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.

Density-independent pixel (dp)

- dp is equivalent to one physical pixel on a 160 dpi screen (medium density screen), the baseline density.
- At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use.
- The conversion of dp units to screen pixels is simple:
  - px = dp * (dpi / 160).
  - For example, 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 of your UI on screens with different densities.

Range of Screens



- layout
  - xlarge screens are at least 960dp x 720dp
  - large screens are at least 640dp x 480dp
  - normal screens are at least 470dp x 320dp
  - small screens are at least 426dp x 320dp

Best Practices

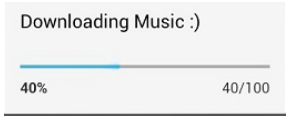
- Use wrap_content, fill_parent, or dp units when specifying dimensions in an XML layout file.
- Do not use hard coded pixel values in your application code.
- Do not use AbsoluteLayout (it's deprecated).
- Supply alternative bitmap drawables for different screen densities.

Further Readings

ProgressDialog


```
Add this as a class variable
private ProgressDialog progress;

In onCreate() add this.
progress = new ProgressDialog(this);
```

- Two types of ProgressDialog.
  - **HORIZONTAL** / SPINNER




```
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progress.show();

final int total = 100;
new Thread() {
    @Override
    public void run() {
        int i = 0;
        while( i < total ) {
           try {
               sleep(100);
           } catch (InterruptedException e) {
               e.printStackTrace();
          }
          i ++;
          progress.setProgress(i);
        }
    }
}.start();
```

Pickers - DatePickerDialog

- To use DatePickerDialog using DialogFragment, you need to define a fragment class that extends DialogFragment and return a DatePickerDialog from the fragement's onCreateDialog() method.
- Create a new class.

```
public class TimePickerFragment extends DialogFragment
                            implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```


Pickers - DatePickerDialog

- Create an instance of TimePickerFragment and call .show() method of the instance.
- This method must be called to show the TimePickerDialog.

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getFragmentManager(), "timePicker");
}
```


Pickers - TimePickerDialog

- Pretty much same as `DatePickerDialog`.

Questions?