CSC 472 / 372 Mobile Application Development for Android

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

Outline

- Android app fundamentals
- Activities
- Simple UI and event handling
- Using resources

Android App Fundamentals

Android App

- A program focusing on performing a small set of related, highly cohesive tasks.
  - Small, focused, cohesive
- The smallest unit that can be packaged, installed, and distributed on Android platforms
  - Presented on the device home screen.
  - Launched by the user
- Packaged into an *Android Package*, APK
  - An archive file with suffix: **.apk**
  - Include compiled code, resources, and data

App Security Sandbox

- Each Android app lives in its own security sandbox
- The Android OS is a multi-user Linux system
  - Each app is a different user
  - All files in an app are only accessible by the app itself
- Each app runs in its own process.
  - Each process has its own virtual machine (VM)
  - Each app runs in isolation from other apps
- There are ways for an app to share data with other apps and for an app to access system services
  - Request and grant permissions

App Components

- Building blocks of an Android app
- Each component is an entry point to an app
  - Entry points for system, not necessary accessible by user
- Four types of app components
  - *Activity* – a single screen of UI
  - *Service* – runs in the background, no UI
  - *Content provider* – manages shared data, no UI
  - *Broadcast receiver* – responds to system-wide broadcast, may create notifications

Our main focus

Activating Components

- An app may activate the components of another app
  - A collaborative eco-system
  - Each app is designed to be a specialist
- An app sends an *intent* to the system to activate a component of an app
- The system activates the component, if the app has the permission
- Components of different apps run in different processes
  - The process of an app is started, if not already running, when a component of the app is activated

Stay tuned

Activities

- An *activity* controls a UI
  - Usually occupy a single screen, or a part of a screen
- Implemented as a subclass of
  - `android.app.Activity`
- The UI is represented as a hierarchy of *view* objects
  - The root of the view hierarchy: *content view* of the activity
    - `setContentView()`
- An app usually consists of multiple loosely coupled activities
  - One is designated as the *main* activity – the launch point

Next topic

Activity Lifecycle

- An activity is one of the smallest unit that can be independently created, paused, resumed, and destroyed
  - Finer granularity than an app
- The life cycle of an activity is managed by the system
- Activities are notified of the life cycle events through *callback* methods

`onCreate()`
- Required
- Invoked when the activity is created
- Initialize the components of activity, e.g., the view objects

More on activity lifecycle and other callback methods later

Callback Methods

- Callback methods are
  - not invoked within the app that implements the callbacks
  - invoked externally, usually by the system
- The purposes are
  - Notify the app, i.e., the receiver of the callback, of certain events, e.g., user action, lifecycle events, system events
  - Provide the app an opportunity to react to the events, e.g., to save or restore app state
- Used extensively in the Android framework

Android App vs. Conventional Program

- Android OS is designed to maximize
  - app performance
  - app security
  - efficiency of resource utilization
- An app's lifecycle is managed by the system
  - An app remains *alive,* even when it is not *running*
    - Fast restart, but retain system resources
  - An app maybe killed by the system, due to low availability of resources

Android App vs. Conventional Program

- An app is not monolithic. A program is.
  - Each app component can be created and destroyed independently
- An app may have multiple entry points. A program usually has one.
- An app may start a component of another app
  - The component runs in *the process of the other app*
  - When a program calls a function, the function runs in *the process of the caller*

Android User Interface Framework

17

Building Blocks of User Interface

- *View* objects – UI components
  - occupies a rectangular area on the screen
  - responsible for drawing and handling events
- The *View* class
  - `android.view.View`
  - Base class for all UI components
- Two types of UI components
  - Widgets – simple elements for user interactions
    - e.g., *Text View*, *Button*
  - *View Groups* – containers to hold other view objects

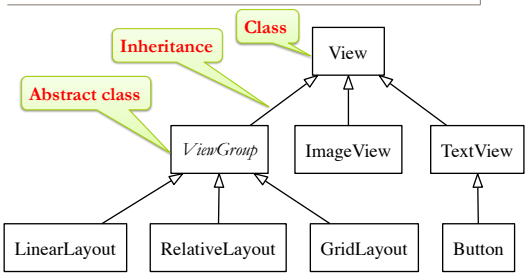
DePaul University 18

View Groups

- *View Group* objects are containers that contain other *View* or *View Group* objects
- The *View Group* class
  - `android.view.ViewGroup`
  - Base class for various *Layouts*
    - e.g., *Linear Layout*, *Relative Layout*
- Each *Layout* class defines the policies and algorithm to spatially arrange its children, i.e., the view objects contained within the layout

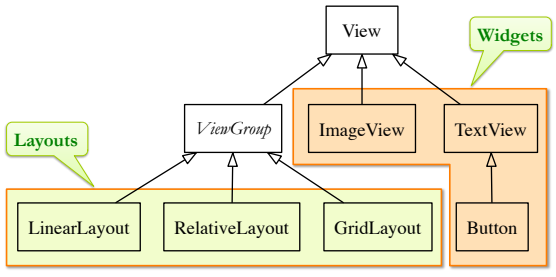
DePaul University 19

Views and View Groups – Inheritance Relation



DePaul University 23

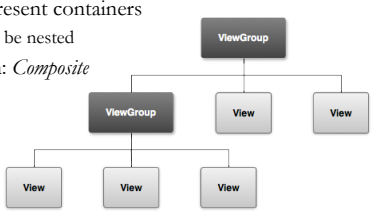
Views and View Groups – Inheritance Relation



DePaul University 25

User Interface View Hierarchy – Ownership Relation

- A view hierarchy consists of *Views* and *View Groups*
- *Views* represent widgets
- *View Groups* represent containers
  - *View Groups* can be nested
- **Design Pattern**: *Composite*



DePaul University 26

Hello Activity!

Hello Activity! – A Single Activity App

- Create a new project with a blank activity
- Edit the `onCreate()` callback method of the activity
  - Create a *Text View* object
    - A static text label
  - Set the content view of the activity
    `void setContentView(View view)`
    - For a view created programmatically
    `void setContentView(int layoutResID)`
    - For a view created from layout resource

The Activity Class MainActivity.java

```
import …

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Superclass of activities

Lifecycle callback

An annotation – this method overrides the same method in the superclass.

The Activity Class MainActivity.java

```
import …

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    …
}
```

Initialize the view of this activity

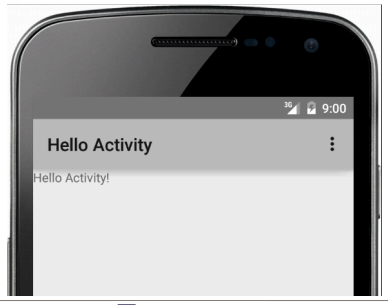
Default behavior defined in the superclass

The Activity Class MainActivity.java

```
import …
import android.widget.TextView;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        TextView tv = new TextView(this);
        tv.setText("Hello Activity!");
        setContentView(tv);
    }
    …
}
```

Create a *Text View* Set the content view

Hello Activity!



Hello Activity! Using Layouts

Layout Classes

- *Layout* classes are subclasses of *View Group*
  - Containers that contain widgets and other nested *Layouts*
  - *View Group* is an abstract class
  - Common *Layouts* include
    - *Linear Layout*, *Relative Layout*, *Grid Layout*, etc.
- Each *Layout* class defines the policies and algorithm to spatially arrange its children, i.e., view objects
  - Constraint-based approach to accommodate different screen sizes
  - Absolute locations and sizes are discouraged

The Linear Layout

- A view group that aligns all children in a single direction
  - vertically, or
  - horizontally
- All children are stacked one after another
  - vertical list, one child per row
  - horizontal list, one child per column
- *Linear Layouts* maybe nested

Hello Activity! – Using Linear Layout

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    LinearLayout layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);

    TextView tv = new TextView(this);
    tv.setText("Hello Activity!");
    layout.addView(tv);

    Button b1 = new Button(this);
    b1.setText("Button 1");
    layout.addView(b1);

    setContentView(layout);
}
```

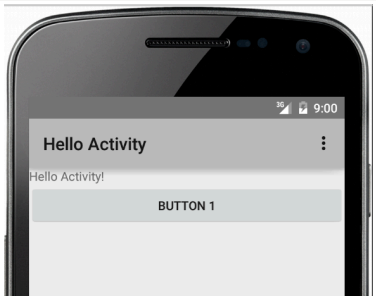
Create a *Layout*

Create a *Text View*

Create a *Button*

Set content view

Hello Activity & Layout!



Hello Activity! Handle UI Events

Hello Activity! – Handle UI Events

```
protected void onCreate(Bundle savedInstanceState) {
    …
    final TextView tv = new TextView(this);
    …
    Button b1 = new Button(this);
    …
    setContentView(layout);

    b1.setOnClickListener(new View.OnClickListener () {
        public void onClick (View v) {
            tv.setText("Button 1 pressed");
        }
    });
}
```

Referenced in the inner class Must be final

An event listener. Anonymous inner class

Interface View.OnClickListener

- An event listener *interface* defined inside the View class
  - `android.view.View.OnClickListener`
- Defines a single *callback* method (abstract)
  `void onClick(View v)`
  - Called when a view is clicked
  - A callback method triggered by a user event

Inner Class

- An *inner class* is a class defined inside another class, known as the *host* class or *outer* class
  - Generally, should not nest more than one level
  - Typically for small helper classes, such as event listeners
- An inner class has access to
  - members declared in the outer class, and
  - *final* variables declared in the enclosing method
- If an inner class is used for creating a single instance, it can be abbreviated to an *anonymous inner class*.

Event Listener as an Inner Class

```
protected void onCreate(Bundle savedInstanceState) {
    …
    final TextView tv = new TextView(this);
    …
    Button b1 = new Button(this);
    …
    setContentView(layout);

    class MyListener implements View.OnClickListener {
        public void onClick (View v) {
            tv.setText("Button 1 pressed");
        }
    }
    b1.setOnClickListener(new MyListener());
}
```

An event listener declared as an inner class

Create an instance of the inner class

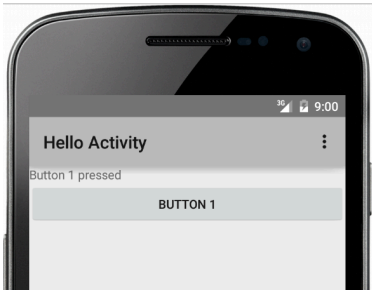
Event Listener as an Anonymous Inner Class

```
protected void onCreate(Bundle savedInstanceState) {
    …
    final TextView tv = new TextView(this);
    …
    Button b1 = new Button(this);
    …
    setContentView(layout);

    b1.setOnClickListener(new View.OnClickListener () {
        public void onClick (View v) {
            tv.setText("Button 1 pressed");
        }
    });
}
```

An event listener declared an anonymous inner class

Hello Activity & Events!



IntelliJ Code Folding

- Why is my listener code look like this?

```
b1.setOnClickListener((v) → { tv.setText("Button 1 pressed"); });
```

- It is a feature – *code folding*
  - Anonymous inner class is folded using the *lambda* syntax
  - Underlying code is unmodified, and can be unfolded

```
b1.setOnClickListener(new View.OnClickListener () {
  public void onClick (View v) {
    tv.setText("Button 1 pressed");
  }
});
```


Using Resources

Android Resources

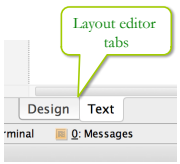
- An Android app typically consists of
  - Java code – to express application logic
  - Resources – to supply data and other contents used in the app, often static
- Resources can be in many different forms
  - Many resources can be defined in XML
- **Best practice**: externalize resources from code
  - Separate static contents from app logic, e.g., layout
  - Accommodate different configurations, e.g., screen sizes
  - Support localization, different languages, e.g., French, Chinese

Organization of Resources

- Many types of resources
  - Layout, string, menu, drawable, animation, style, etc.
- Resources are located in subfolders under **res**
- Organized according to
  - Types, e.g., layout, string, etc. and
  - Configurations, e.g., default, alternatives for different screen sizes, locales etc.
- A resource may have a different value for each different configuration
- Resources are managed by the system

Android Resources – Layout

- Define the composition of UI in XML
  - The layout of a UI screen (of an activity), or
  - A component of the UI, which can be used elsewhere
- Stored in the res/layout/ folder
- The layout resources can be edited
  - *graphically* (using the "Design" tab), or
  - *textually* (using the "Text" tab)
- We will use the textual form



MVC Architecture

- Principle of *Separation of Concerns*
- Organization of the modules according to their roles and responsibilities:
  - *Models* – responsible for managing data
  - *Views* – responsible for visual representations
  - *Controllers* – responsible for interaction and coordination
- Separation of the presentation from the behavior
  - Layout resources: manage the presentation, i.e., the *Views*
  - Activities: manage the behavior, i.e., the *Controllers*

Layout Resources

- A layout resource defines a view hierarchy

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup
      xmlns:android="http://schemas.android.com/apk/res/android"
      [ViewGroup attributes] >
    <View [View attributes] >
    </View>
    …
    <ViewGroup [ViewGroup attributes] >
       <View [View attributes] />
       …
    </ViewGroup>
</ViewGroup>
```


Elements in Layout Resource – Correspondence with a View Class

- Elements `<ViewGroup>` and `<View>` are placeholders
- Each `<ViewGroup>` element corresponds to a subclass of *View Group* class, i.e., a container
  - e.g., `<LinearLayout>` (java.widget.LinearLayout), `<RelativeLayout>` (java.widget.RelativeLayout)
- Each `<View>` element corresponds to a widget subclass of *View* class, i.e., a non-container, widget
  - e.g., `<TextView>` (java.widget.LinearLayout), `<Button>` (java.widget.Button)

Common Attributes of View Group & View Elements

- Attribute: `android:id`
  - Resource ID. Must be unique within a resource file.
- Values are in the form of `"@+id/name"`
  - `@` sign: indicates a resource string
  - `+` sign: indicates a new resource name

Common Attributes of View Group & View Elements

- Attributes: `android:layout_width` and `android:layout_height`
  - The width and height of the element. **Required**.
- Values can be one of the following
  - Dimension values, or
  - One of the following keywords
    `match_parent` a.k.a. `fill_parent` (deprecated)
    - Sets the dimension to match that of the parent element

    `wrap_content`
    - Sets the dimension only to the size required to fit the content

Specify Dimensions in Resources

- Fixed units
  - Units based on the physical size of the screen
  - Not recommended. Should be avoided
- Supported fixed units
  - `pt` – points, 1/72 of an inch
  - `px` – pixels on the screen.
    - Screen densities (pixels per inch) vary
  - `mm` – millimeters
  - `in` – inches

Specify Dimensions in Resources

- Scalable units
  - Automatically scaled to maintain consistent size across different devices
- Supported scalable units
  - `dp` (or `dip`) – density-independent pixels.
    - 1dp ≈ 1px on 160 dpi screen
    - Proportionally scaled for screens with higher dpi
    - Recommended for specifying UI element sizes
  - `sp` – scale-independent pixels, 1sp ≈ 1dp
    - Scaled based on the screen density and the user's font size preference
    - Recommended for specifying font sizes

Layout Element-Specific Attributes

- Each layout element defines a set of element-specific attributes
- These attributes correspond to the properties defined in the corresponding *View* or *View Group* class
  - XML attribute of a layout element:
    - `android:attribute`
  - Java property of the corresponding *View* class:
    - `getAttribute()`
    - `setAttribute(v)`

DePaul University 79

Hello Resources!

80

Hello Resources!

- A simple app with a UI created from layout resource
- Create an app with a blank activity
- Edit the automatically generate layout resource file **res/layout/activity_main.xml**
  - Remove the *Relative Layout* and the *Text View* element
  - Replace the top element with a *Linear Layout*
  - Add three child elements to the top layout: a *Text View* and two *Buttons*
- Leave the activity file (.java) unchanged

DePaul University 81

Define UI as Layout Resource activity_main.xml

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

  <TextView
    android:id="@+id/text1"
    android:text="Hello Resources!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:textStyle="bold" />
```

DePaul University 82

Define UI as Layout Resource activity_main.xml (cont'd)

```
  <Button
    android:id="@+id/button1"
    android:text="Button 1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <Button
    android:id="@+id/button2"
    android:text="Button 2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

DePaul University 83

The Activity – The Default Implementation

```
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

}
```

What is `R.layout.activity_main`?
Where is it declared?
What is its type?

A resource ID referring to the layout resource defined in res/layout/activity_main.xml

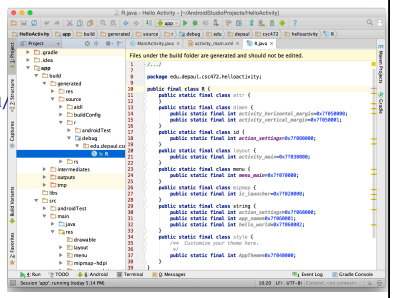
DePaul University 86

Resources and R.java

- Everything defined as a resource in XML is accessible in Java code
- Resources are precompiled
- Each resource has a unique ID for it to be accessed in Java code.
  - ID's for all user defined resources are defined in an automatically generated Java class named *R*
  - R is in the root package of the app
  - Resource ID's are declared as int values
  - R.java is updated whenever resource definitions are changed

The Auto-Generated R.java

- Contains a Java class named **R**
- Located under **app/build/generated/source/r**
  - Intentionally hidden
  - Visible in the *Project* view
- **Do not modify R.java**



Access Resources in Code

- Access layout resource
  - View hierarchy defined in layout file

    res/layout/*activity_name*.xml
  - `R.layout.activity_name`
- Access view objects with id of "*view_id*", i.e., the value of `android:id` attribute is "*view_id*"
  - `R.id.view_id`
- In general, for a resource of type *res_type* and id of "*res_id*"
  - `R.res_type.res_id`

The Activity – The Default Implementation

```
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

}
```

`R.layout.activity_main` is a constant defined in R.java of type int.

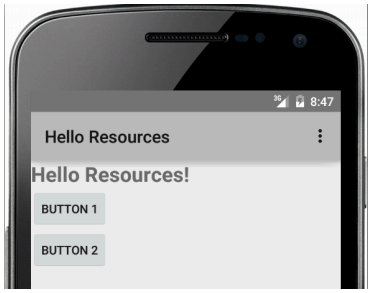
A resource ID referring to the layout resource defined in **res/layout/activity_main.xml**

Layout Inflation

- Set content view method

  `void setContentView(int layoutResID)`
  - For a view created from layout resource
- What happens when the method is called?
  - Parse the layout resource file (XML) referenced by `layoutResID`
  - *Inflate* the layout resource, i.e., construct the view hierarchy described in the layout resource
  - Set the inflated view hierarchy as the content view of the activity

Hello Resources! UI Defined in Layout Resource



Define String Resources

- Define string constants as resources in res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Hello Resources</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>

  <string name="hello">Hello Resources!</string>
  <string name="button1">Button 1</string>
  <string name="button2">Button 2</string>
</resources>
```


Reference String Resources

- String resource definitions

```
<resources>
    <string name="key1">String value 1</string>
    <string name="key2">String value 2</string>
    …
</resources>
```

- Reference from other resources (XML), e.g., layout
  `@string/key1`
  `@string/key2`
- Reference from code (Java), e.g., activity
  `R.string.key1`
  `R.string.key2`

Reference String Resources in Layout

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

  <TextView
    android:id="@+id/text1"
    android:text="@string/hello"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:textStyle="bold" />
```


Reference String Resources in Layout (cont'd)

```
  <Button
    android:id="@+id/button1"
    android:text="@string/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <Button
    android:id="@+id/button2"
    android:text="@string/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```


Hello Resources! Handle UI Events

Find View by ID

- A method of the Activity class
  `public View findViewById (int view_id)`
  - Finds the view whose id is *view_id*
- The layout resource must be inflated, using `setContentView()`, before this method is called.
- The *view_id* is available from class R, if the `android:id` attribute of the view is *"view_id"*
  - `R.id.view_id`

Reference UI Widgets – Layout Resource: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout … >

  <TextView
    android:id="@+id/text1"
    … />
  <Button
    android:id="@+id/button1"
    … />
  <Button
    android:id="@+id/button2"
    … />

</LinearLayout>
```

DePaul University 102

Reference UI Widgets – Activity: MainActivity.java

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  final TextView tv = (TextView) findViewById(R.id.text1);
  Button b1 = (Button) findViewById(R.id.button1);
  Button b2 = (Button) findViewById(R.id.button2);

  …
}
```

Need to downcast.

Find the view objects defined in res/layout/activity_main.xml

DePaul University 106

Handle UI Events – Activity: MainActivity.java (cont'd)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  …
  View.OnClickListener listener =
    new View.OnClickListener () {
      public void onClick (View v) {
        tv.setText(((Button) v).getText() + " pressed");
      }
    };
}
```

Need to downcast.

The view object *v* is the widget that triggered the event

An event listener as an anonymous inner class

DePaul University 111

Handle UI Events – Activity: MainActivity.java (cont'd)

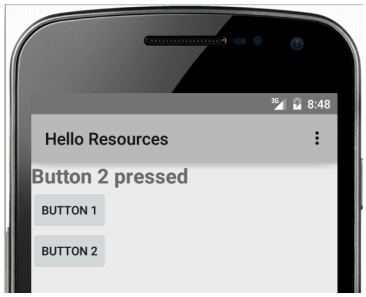
```
@Override
protected void onCreate(Bundle savedInstanceState) {
  …
  View.OnClickListener listener =
    new View.OnClickListener () {
      public void onClick (View v) {
        tv.setText(((Button) v).getText() + " pressed");
      }
    };

  b1.setOnClickListener(listener);
  b2.setOnClickListener(listener);
}
```

Both buttons use the same event listener

DePaul University 113

Hello Resources! – Handle Events



DePaul University 116

Hello Resources! More on Linear Layout

Nesting Linear Layouts

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout …
    android:orientation="vertical">
  <TextView … />
  <Button … />
  <Button … />
  <LinearLayout …
      android:orientation="horizontal">
    <Button … />
    <Button … />
  </LinearLayout>
</LinearLayout>
```

The outer layout is vertical

The inner layout is horizontal

DePaul University

Nesting Linear Layouts

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout …
    android:orientation="vertical">
  <TextView … />
  <Button … />
  <Button … />
  <LinearLayout …
      android:orientation="horizontal">
    <Button … />
    <Button … />
  </LinearLayout>
</LinearLayout>
```

DePaul University

Adjust the Margins

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout …
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical">
…
</LinearLayout>
```

Specify the margin on each side

DePaul University 121

Adjust the Margins

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout …
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical">
…
</LinearLayout>
```

DePaul University 122

The Sample Code

- The sample apps in this lecture are available in D2L
  - HelloActivity.zip
  - HelloResources.zip
- Each zip archive contains the entire project folder
- Download and unzip the files on your local machine
- From Android Studio:
  - File | Import Project …
  - Choose the folder

Important: Do not use File | Open …

DePaul University 124

Next …

- Layout parameters
- Relative layout
- More widgets
  - Text fields
  - Radio buttons, checkboxes, toggle buttons
  - Sliders
  - Images

DePaul University 125