Week 3: UI Design and Layouts

UW PCE Android Application Development Program Course 1 – Android Development Fundamentals

Agenda

- UI Intro review
- User Interface Layouts
  - ListView
  - GridView
  - Homework 2

UI Intro - review

- Layouts
  - Defines the visual structure for a UI, such as Activity or App Widget.
  - Two ways to declare layouts:
    - Declare UI elements in XML
    - Instantiate layout elements at runtime.
  - XML declaration separates presentation from control.

- XML
  - Must contain exactly one root element, which must be a View or ViewGroup object
  - Can add several layouts as child elements to build up the view hierarchy

UI Intro review: XML example

```
<?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/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
   <Button android:id="@+id/button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Hello, I am a Button" />
</LinearLayout>
```

UI Intro review: Load XML resource

- On compilation, XML layout file is compiled into a view resource.
- Layout resource should be loaded from Activity.onCreate() callback
  - onCreate is called on Activity launch
  - Call setContentView(), pass in reference to XML resource in the form: R.layout.*layout_file_name*

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main_layout);
}
```

UI Intro review: Layout Attributes

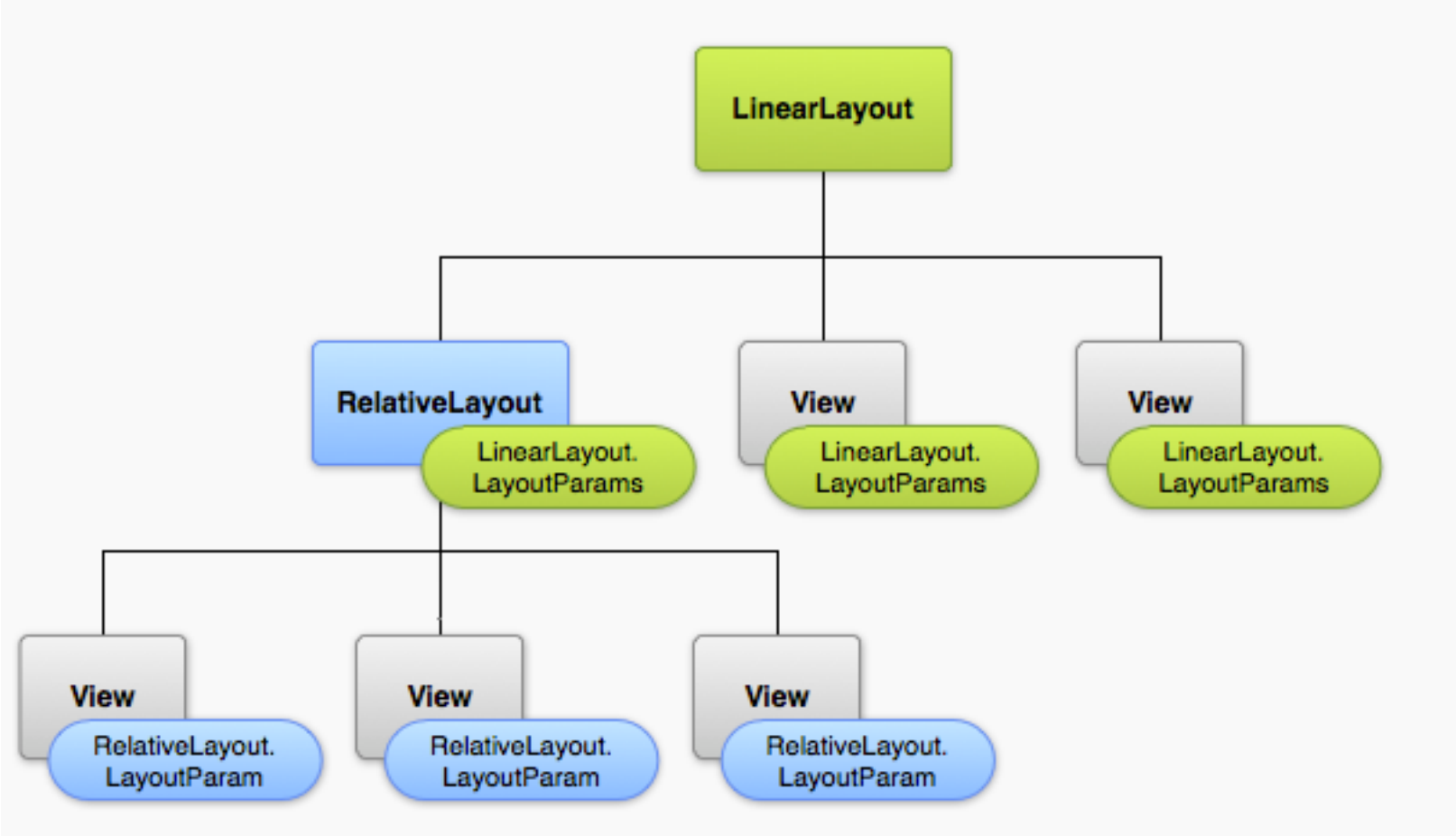
- Every View and ViewGroup support own XML attributes
  - Some attributes are specific to a View object
    - E.g. TextView (and all child objects) supports textSize
  - Some are common to all View objects from inheritance
- ID
  - Any View may have an integer ID assigned to it
  - Common to all View objects
  - Takes the form:
    - android:id="@+id/my_button”
    - “+” means create and add to the R.java resource
  - Android id resource takes the form:
    - android:id="@android:id/empty"

UI Intro review: Layout Parameters

- XML attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which is resides

- Each ViewGroup class implements a nested class that extends ViewGroup.LayoutParams
  - Subclass contains properties for size and position
  - Each ViewGroup is required to define layout_width and layout_height
    - *Exact measurements – you probably won't do this often, but use "dp"*
    - *wrap_content* - size to dimensions required by View
    - *match_parent* - size to dimensions as big as parent ViewGroup will allow

UI Intro review: Layout Parameters



UI Intro review: Layout Position

- The geometry of a View is that of a rectangle
  - Location: expressed as a pair of left and top
  - Dimensions: expressed as width and height
  - Unit for both location and dimension is pixels
  - getLeft() retrieves left, or x coordinate, relative to the parent
  - getTop() retrieves top, or y coordinate, relative to the parent

- Convenience methods
  - getRight() retrieve the right edge of the rectangle representing the view
  - getBottom() retrieves the bottom edge of the rectangle rep. the view
  - i.e. getRight() = getLeft() + getWidth().

UI Intro review: Size, Padding & Margins

- The size of a View is expressed in width and height
  - measuredWidth and measuredHeight
  - getMeasuredWidth() and getMeasuredHeight()

- Padding
  - Offset (in px) from an edge of the dimensions of a View
  - Expressed in left, right, top, bottom
  - Set using setPadding(int,int,int,int)

- Margins
  - Views can define paddings, but cannot define margins
  - ViewGroup provide support for margin

UI Intro - Common Layouts

- Layouts are subclasses of the ViewGroup class
  - Provide unique way to display views nested within

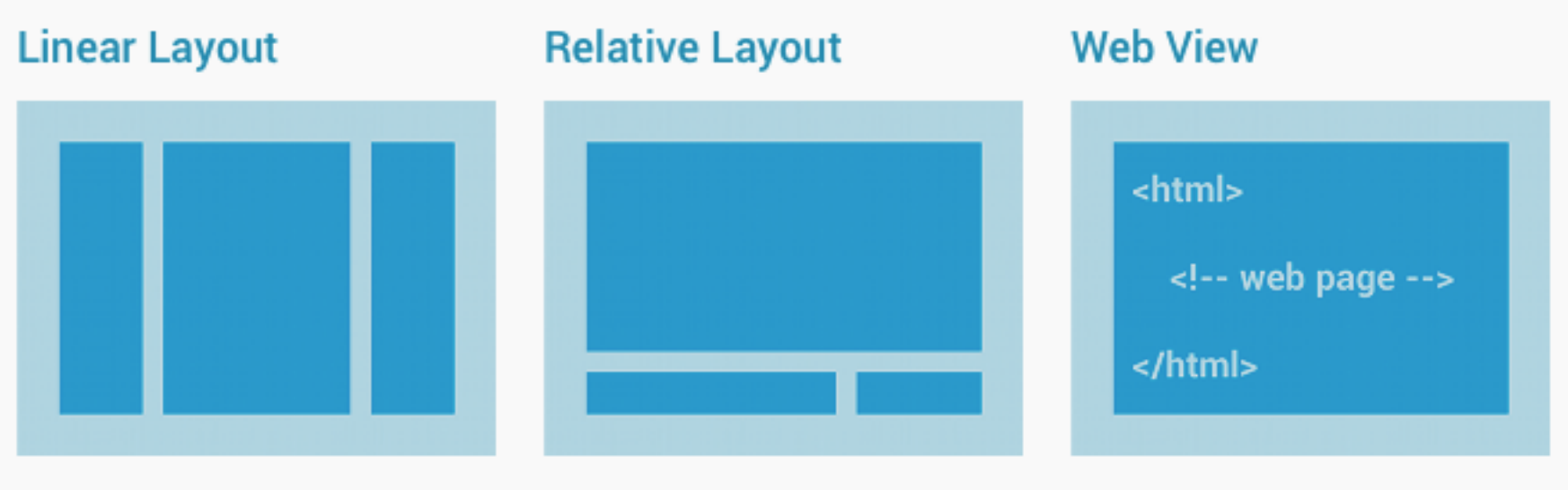
- The Android platform has built in layouts. Some of the commonly used layouts are:
  - LinearLayout
  - RelativeLayout
  - WebView
  - ListView
  - GridLayout

Common Layouts – Linear and Relative

- LinearLayout
  - Children are organized in single horizontal or vertical lines
  - Adds scrollbar is length is greater than length of screen

- RelativeLayout
  - Organizes the children in positions that are relative to each other.

Common Layouts – Linear, Relative & Web



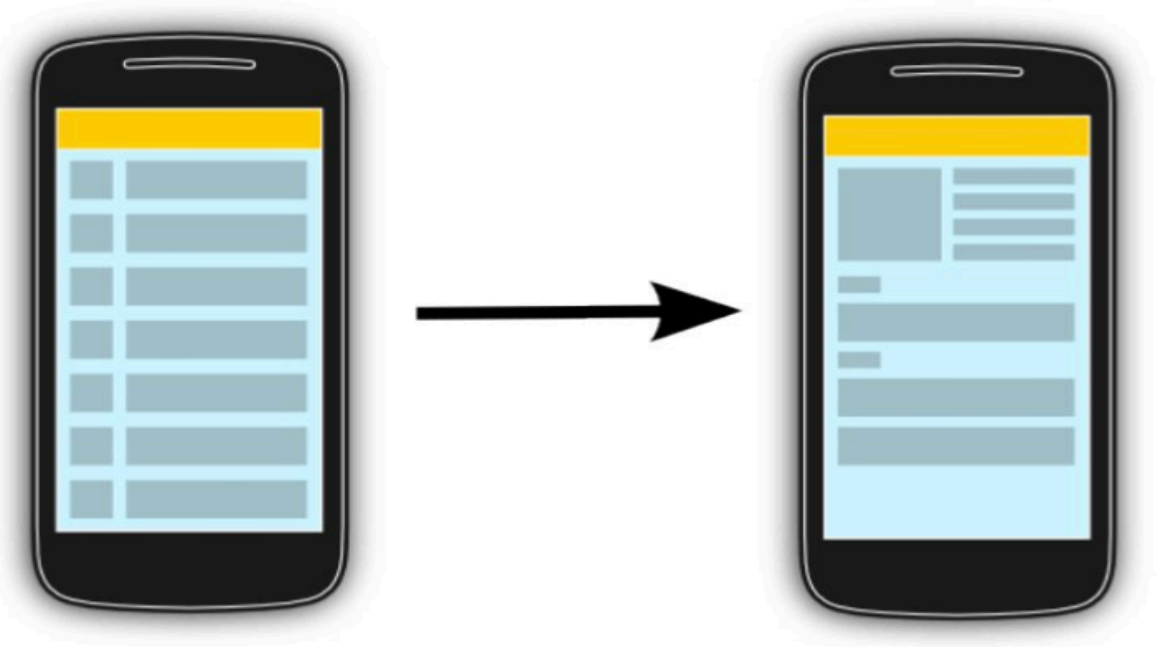
Layouts with Adapter

- Layouts with dynamic contents are usually built by subclassing AdapterView to populate views at runtime
  - Subclassed AdapterView layouts are bound to their data with an Adapter
  - AdapterView extends ViewGroup

- Adapter:
  - a “middleman” between a data source and the AdapterView layout
  - Retrieves the data and converts each entry to a View that can be added to the AdapterView layout
  - Possible data sources are arrays, database query, data from web service, etc.

Layouts with Adapter - ListView

- A ListView displays data in a vertically scrolling pattern.
  - Each item can be handled separately.
  - On selection, each item can start a new Activity



ListView - Adapter

- ListView is populated by binding to an Adapter
  - Adapter retrieves the data from source and creates a View representing each data entry

- Android provides several common Adapter subclasses
  - ArrayAdapter
    - can handle a list or array of Java objects
    - Every object is mapped to a row in the layout
    - Ref: http://developer.android.com/reference/android/widget/ArrayAdapter.html
    - Example – display an array of strings in a ListView

ListView – ArrayAdapter example

- Example - display an array of **strings** in a ListView

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, myStringArray);
```

- Call setAdapter() on your ListView

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

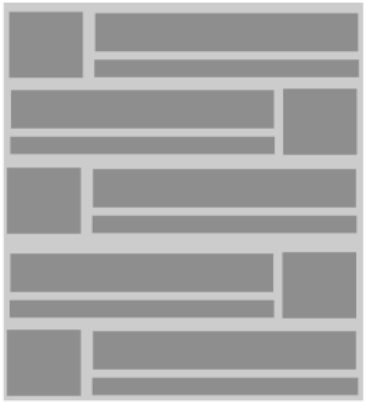
ListView – ArrayAdapter example

- Code samples

ListView – Custom adapters

- ArrayAdapter is limited in scope
  - Supports only mapping of toString() to one view in the row layout
  - In reality, many apps have several views per row
  - To achieve complex layouts per row, write a custom adapter class that extends ArrayAdapter or directly extend BaseAdapter

- For example, consider this ListView



  - Each row has a different layout
  - Even and odd rows have similar layouts
  - Let's build a custom adapter to bind this to the ListView

ListView – Custom adapters

- To write a custom adapter
  - Override the getView() method of the adapter
  - Inflate an XML based layout and set the contents of the individual row based on the Java object
  - To inflate, use the LayoutInflater system service

- LayoutInflater
  - Instantiates an XML into its corresponding View object
  - Never used directly. Get an instance by:

```
LayoutInflater inflater = (LayoutInflater)context.getSystemService
    (Context.LAYOUT_INFLATER_SERVICE);

LayoutInflater inflater  = getLayoutInflater()
```

ListView – Custom adapters & performance

- Default ArrayAdapter is already performance optimized
- In custom adapters, each view is inflated from XML
- Inflating and creating Java objects is time consuming
  - Time and memory consumption
  - findViewById() is relatively time consuming

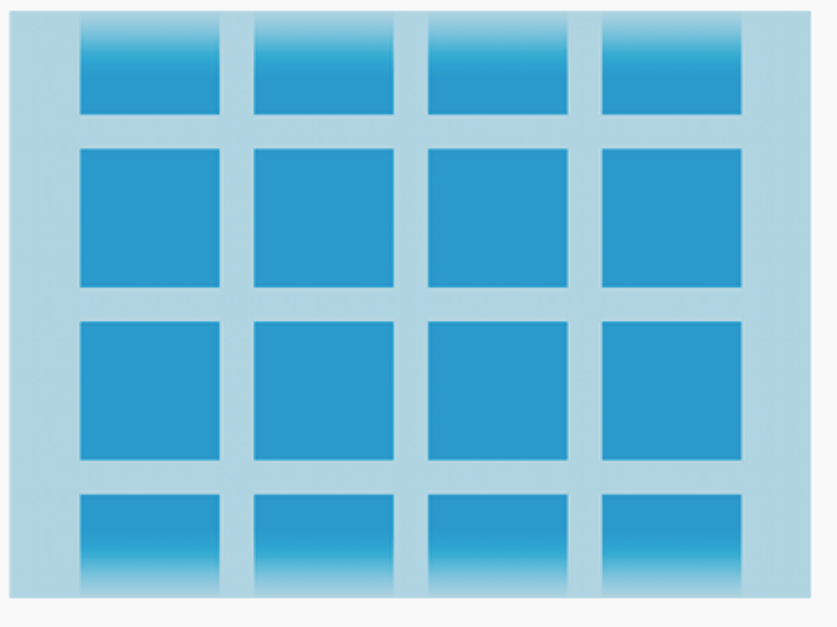
- Use Holder Pattern
  - A View Holder pattern allows to avoid the findViewById()
  - A static inner class inside adapter, which holds references to relevant views
  - Faster than using the findViewById() method.

ListView – Optimized adapter example

- Code example

GridView

- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.
- Grid items are automatically inserted to the layout using a ListAdapter



GridView - example

- Code sample