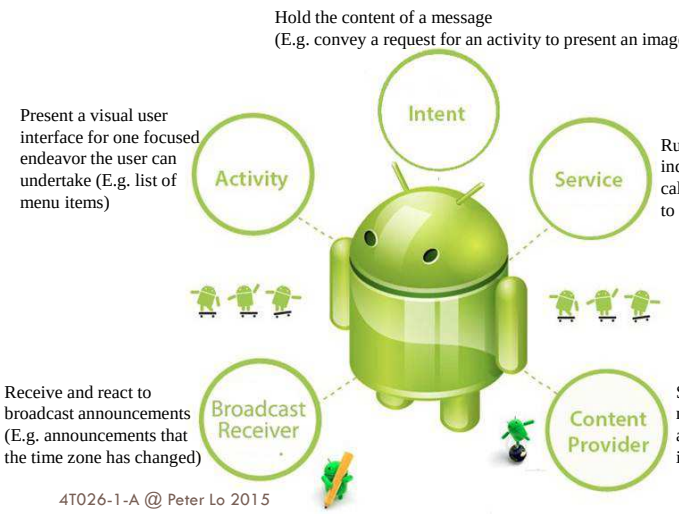
ANDROID APPS DEVELOPMENT FOR MOBILE GAME

Lecture 2: Android Layout and Permission

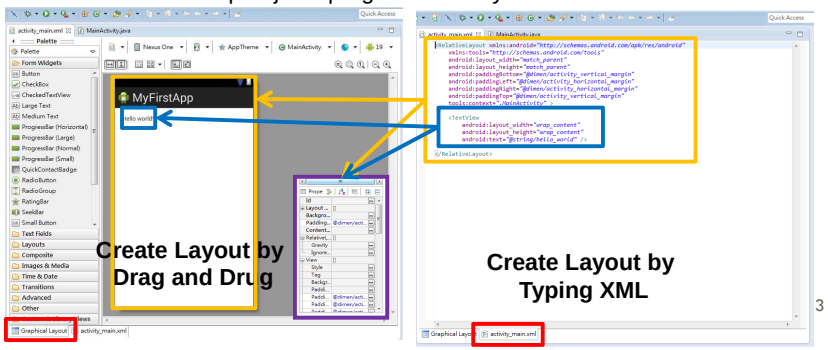
Peter Lo

Application Components



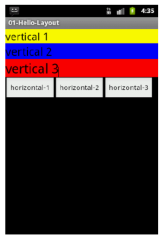
Simple Android App Layout

- A layout defines the visual structure for a user interface:
  - Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - Instantiate layout elements at runtime. Your application can create View and ViewGroup objects programmatically.

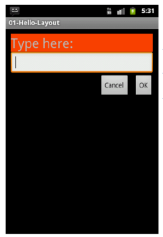


Android Layouts

- An Android layout is a class that handles arranging the way its children appear on the screen.
- Anything that is a View can be a child of a layout.
- All of the layouts inherit from ViewGroup so you can nest layouts.
  - AbsoluteLayout
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout



Linear Layout
A LinearLayout places its inner views either in horizontal or vertical disposition.



Relative Layout
A RelativeLayout is a ViewGroup that allows you to position elements relative to each other.

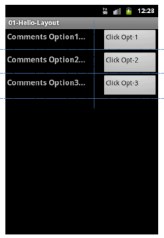
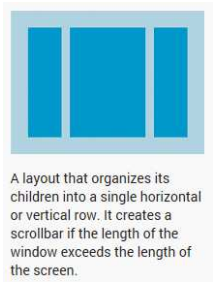
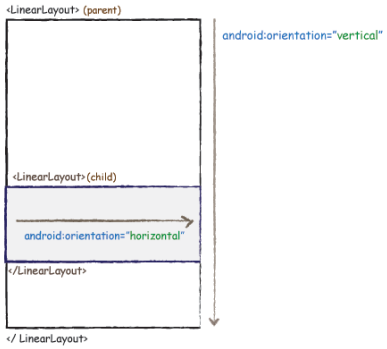


Table Layout
A TableLayout is a ViewGroup that places elements using a row & column disposition.

Linear Layout

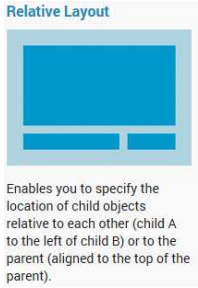
- In a linear layout, all the elements are displayed in a linear fashion, either Horizontally or Vertically
- The properties is set in ***android:orientation*** which is an attribute of the node ***LinearLayout***.





Relative Layout

- In a relative layout every element arranges itself relative to other elements or a parent element.
  - E.g. The “Cancel” button is placed relatively, to the right of the “Login” button parallel. Here is the code snippet that achieves the mentioned alignment



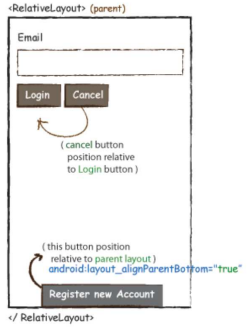
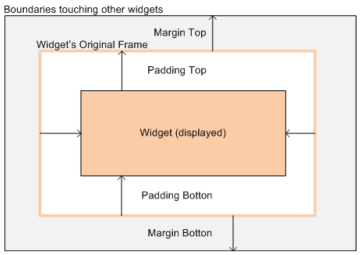
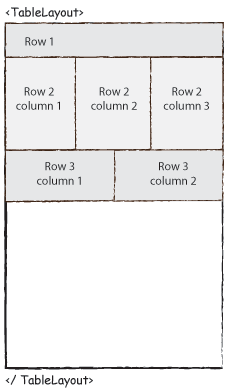


Table Layout

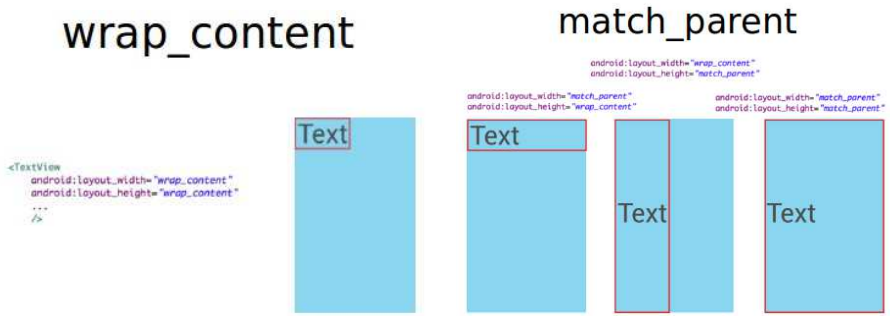
- Table layouts in Android works in the same way HTML table layouts work.
- You can divide your layouts into rows and columns.





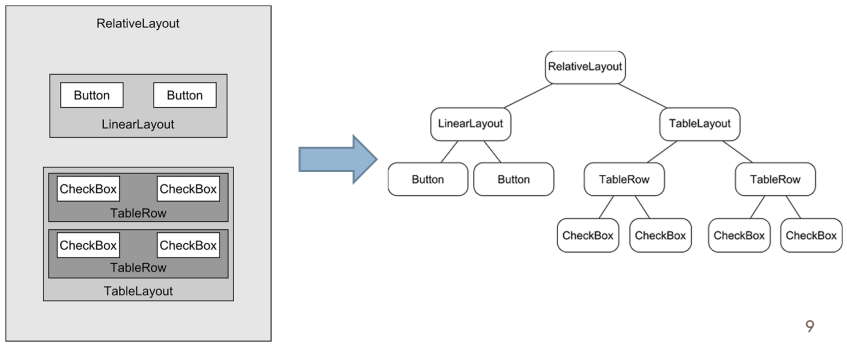
Layout Attributes

- FILL_PARENT / MATCH_PARENT, which means that the view wants to be as big as its parent (minus padding)
- WRAP_CONTENT, which means that the view wants to be just big enough to enclose its content (plus padding)



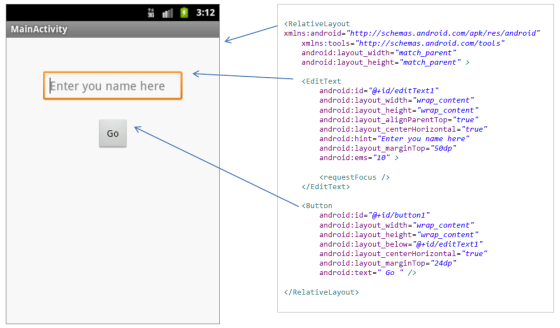
The View Hierarchy

- The view hierarchy diagram gives probably the clearest overview of the relationship between the various views that make up the user interface shown. When a user interface is displayed to the user, the Android runtime walks the view hierarchy, starting at the root view and working down the tree as it renders each view.



Using Views

- Dealing with widgets & layouts typically involves the following operations
  - Set properties
  - Set up listeners
  - Set focus
  - Set visibility

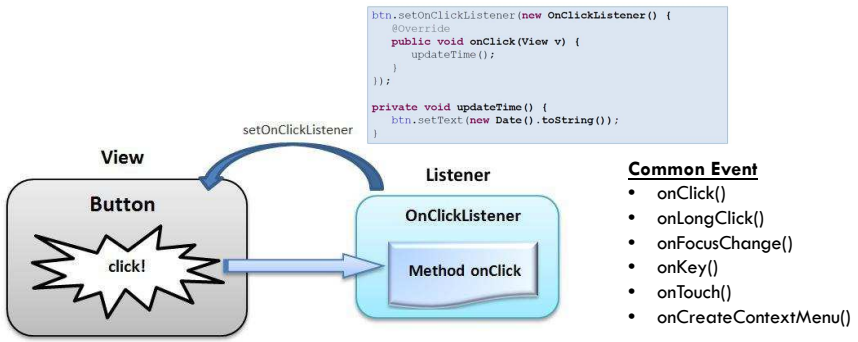


Actual UI displayed by the app

Text version: *activity_main.xml* file

Event Listener

- An object cannot process event on its own, it needs a listener for this.
  - E.g. When a button is clicked, listener reacts and runs the code from **onClick** method.

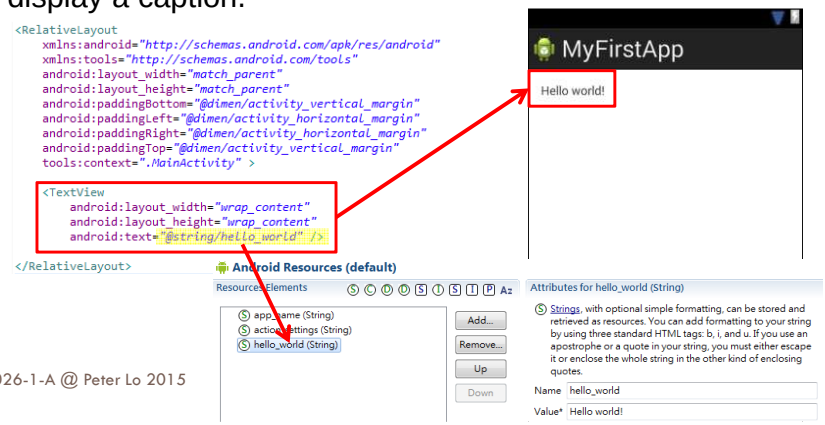


Common Event

- onClick()
- onLongClick()
- onFocusChange()
- onKey()
- onTouch()
- onCreateContextMenu()

TextView (Label)

- A label is called in android a TextView.
- TextViews are not editable, typically used for output to display a caption.



Button

- A Button widget allows the simulation of a clicking action on a GUI. Button is a subclass of TextView, it styled using the system's default button background.

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

Request a reference to the button from the activity by calling *findViewById*.

Create a class implementing "OnClickListener" and set it as the on click listener for the button.

EditText (Text Field)

- EditText allows the user to type text (single text or multi-line) into your app.
- Touching a text field places the cursor and automatically displays the keyboard.

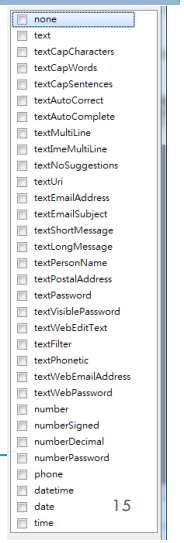
```
public class MainActivity extends Activity {
    EditText UserInput;

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

        UserInput = (EditText) findViewById(R.id.editText1);
        UserInput.setOnFocusChangeListener(new OnFocusChangeListener() {
            public void onFocusChange(View v, boolean hasFocus) {
                String strValue = UserInput.getText().toString();
            }
        });
    }
}
```

Request a reference to the button from the activity by calling *findViewById*.

Retrieve the EditText value and convert to string

Common InputType for EditText

- text — hello
- number — 123456
- phone — (852) 2766-5111
- textMultiLine — first row / second row
- textCapCharacters — HELLO
- textPassword — ••••••••d
- textAutoCorrect — teh ⇒ the



Radio Button

- Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side.
- To create each radio button option, create a ***RadioButton*** in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a ***RadioGroup***. By grouping them together, the system ensures that only one radio button can be selected at a time

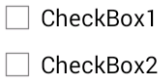
(●) RadioButton1
( ) RadioButton2
( ) RadioButton3

Handling Events in Radio Button

```
RadioButton radio0 = (RadioButton) findViewById(R.id.radio0);
RadioButton radio1 = (RadioButton) findViewById(R.id.radio1);

RadioGroup radioGroup1 = (RadioGroup) findViewById(R.id.radioGroup1);
radioGroup1.setOnCheckedChangeListener( new RadioGroup.OnCheckedChangeListener() {
        public void onCheckedChanged ( RadioGroup group, int checkedId ) {
                if ( checkedId == radio0.getId() ) {
                        Your Action ...
                } else if ( checkedId == radio1.getId() ) {
                        Your Action ...
                }
        }
} );
```


Checkbox

- Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.
- To create each checkbox option, create a ***CheckBox*** in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.



Handling Events in Checkbox

```
CheckBox checkbox1 = (CheckBox) findViewById(R.id.checkBox1);
if (checkbox1.isChecked()) {
                Your Action ...
} else {
                Your Action ...
}

CheckBox checkbox2 = (CheckBox) findViewById(R.id.checkBox2);
if (checkbox2.isChecked()) {
                Your Action ...
} else {
                Your Action ...
}
```


Toast (Message)

- Toast messages is a simple pop-up message for alerting users.
- The following code would generate a toast message:
  - ***Toast.makeText(Context, Text, Duration).show();***

The context to use, usually your Application or Activity object by method *getApplicationContext()*

The text to show on screen

Duration for the message display:
- *LENGTH_SHORT* (2 seconds)
- *LENGTH_LONG* (3.5 seconds)

Using Permissions

- A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely impact the user experience or any data on the device.
- To make use of protected features of the device, you must include in your AndroidManifest.xml one or more <uses-permission> tags declaring the permissions that your application needs

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

- The permissions provided by the Android system can be found at
  - http://developer.android.com/reference/android/Manifest.permission.html

Common Permission

Permission	Description
ACCESS_FINE_LOCATION	Allows an app to access precise location from location sources such as GPS, cell towers, and Wi-Fi.
BLUETOOTH	Allows applications to connect to paired Bluetooth devices
CAMERA	Required to be able to access the camera device.
INTERNET	Allows applications to open network sockets.
READ_CONTACTS	Allows an application to read the user's contacts data.
READ_EXTERNAL_STORAGE	Allows an application to read from external storage.
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage.
NFC	Allows applications to perform I/O operations over NFC
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed.