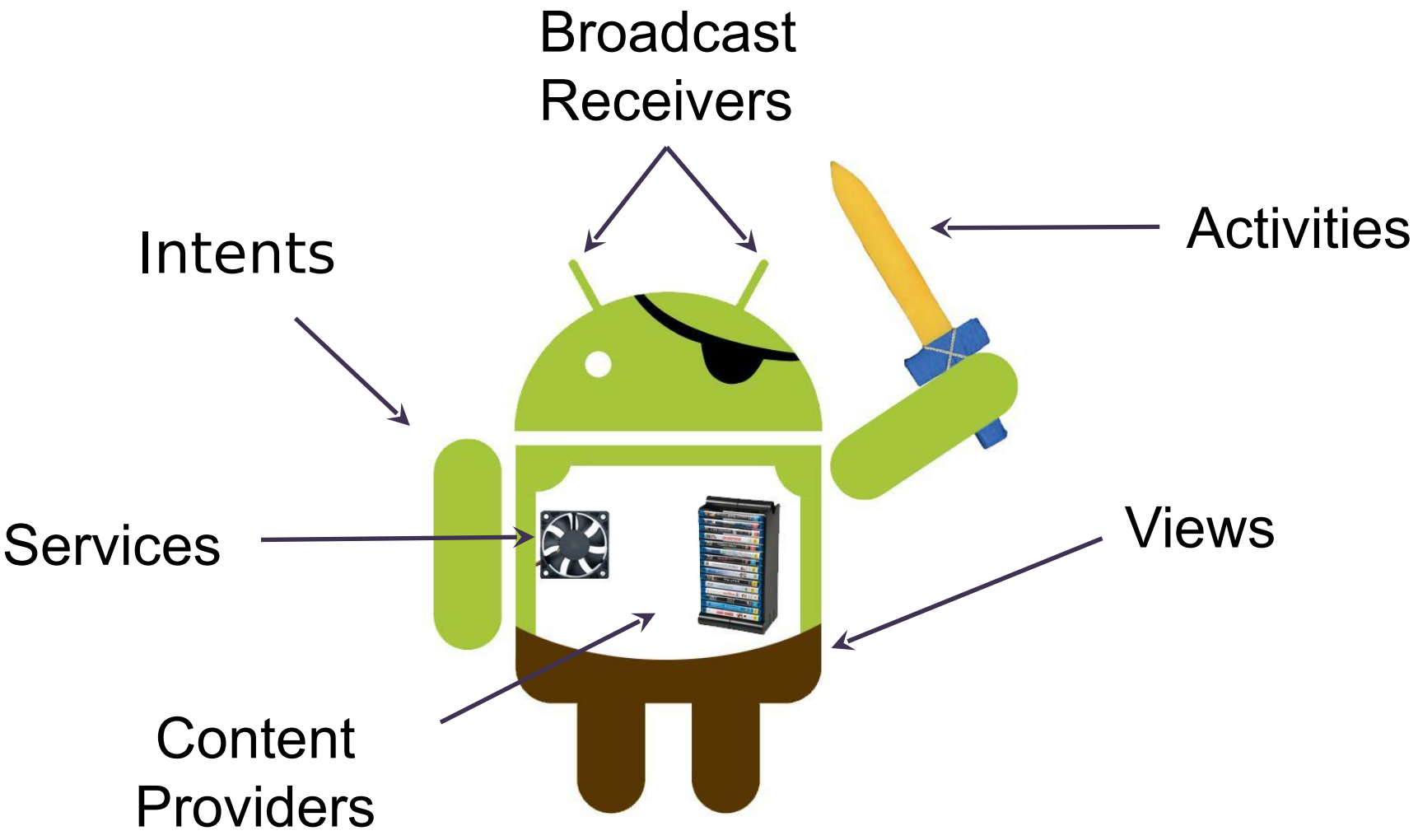
Android Application User Interface

Anatomy of Android application



Application building blocks

- **AndroidManifest**
  - general configuration for your application
  - contains all component definitions of your appication

- **Activities***
  - application presentation layer
  - single, focused things that user can do
  - kind of “screen”
  - layout and display information with Fragments and Views
  - respond to user actions
  - well-defined lifecycle

Application building blocks

- **Services**
  - run without UI
  - update data sources and Activities
  - trigger Notifications
  - used to perform long running tasks w/o user interaction

- **Content Providers**
  - shareable persistent data storage
  - across application boundaries
  - e.g. Contacts, MediaStore, Settings

Application building blocks

- **Intents**
  - interapplication message-passing framework
  - used to request an action be performed on a particular piece of data
  - kind of absolute and relative links

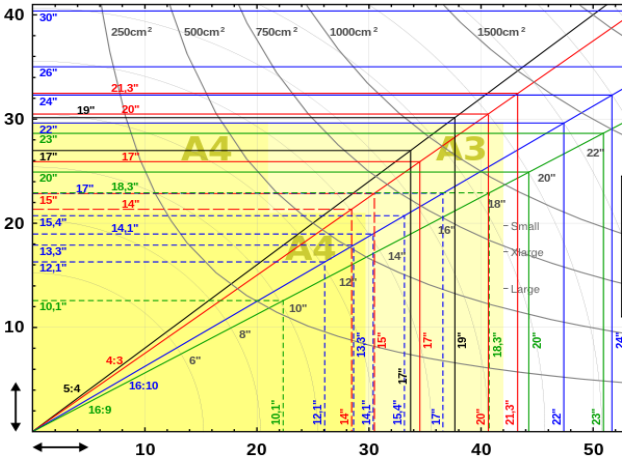
- **Broadcast Receivers**
  - listen for Intents that match the criteria
  - e.g. SMS message

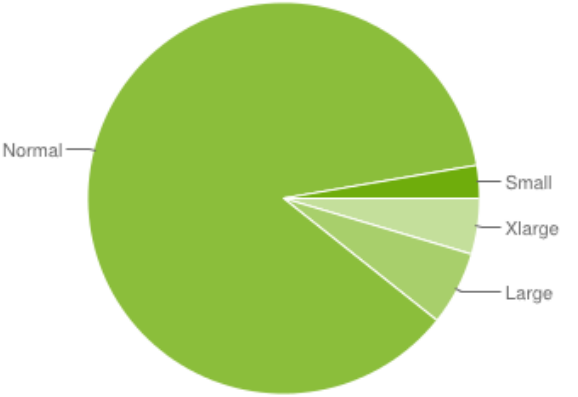
- **App Widgets**
  - Visual application components that are added to the home screen

Android UI fundamentals

Android UI fundamentals

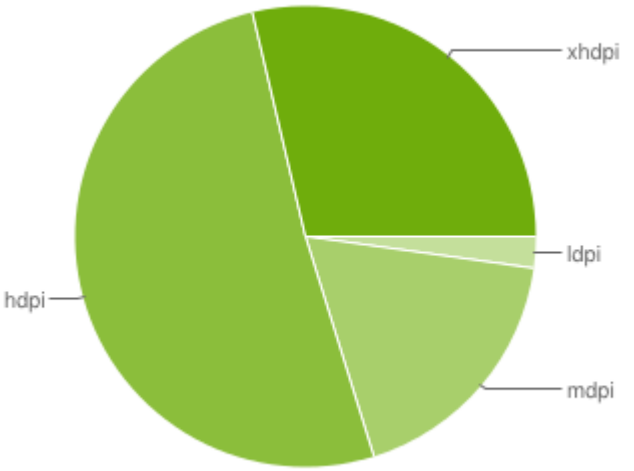
- **Screen size**
  - actual physical size, measured as the screen's diagonal
  - 2.55”, **3.2”**, 4.0”, 10.1”, ...
  - Android groups into four generalized sizes:
    - Small
    - Normal
    - Large
    - Xlarge





Android UI fundamentals

- **Screen density**
  - quantity of pixels within a physical area of the screen (dots per inch, dpi).
  - "low" density screen has fewer pixels
  - Android groups actual screen densities into four generalized densities:
    - Low (ldpi, 120)
    - Medium (mdpi, 160)
    - High (hdpi, 240)
    - Extra high (xhdpi, 320)



	ldpi	mdpi	hdpi	xhdpi
small	1.7%		1.0%	
normal	0.4%	11%	50.1%	25.1%
large	0.1%	2.4%		3.6%
xlarge		4.6%		

Android UI fundamentals

- **Density-independent pixel (dp)**
  - virtual pixel unit to express layout dimensions or position in a density-independent way
  - equivalent to one physical pixel on a 160 dpi screen
  - px = dp * (dpi / 160)
  - system transparently handles any scaling of the dp units

Android UI fundamentals

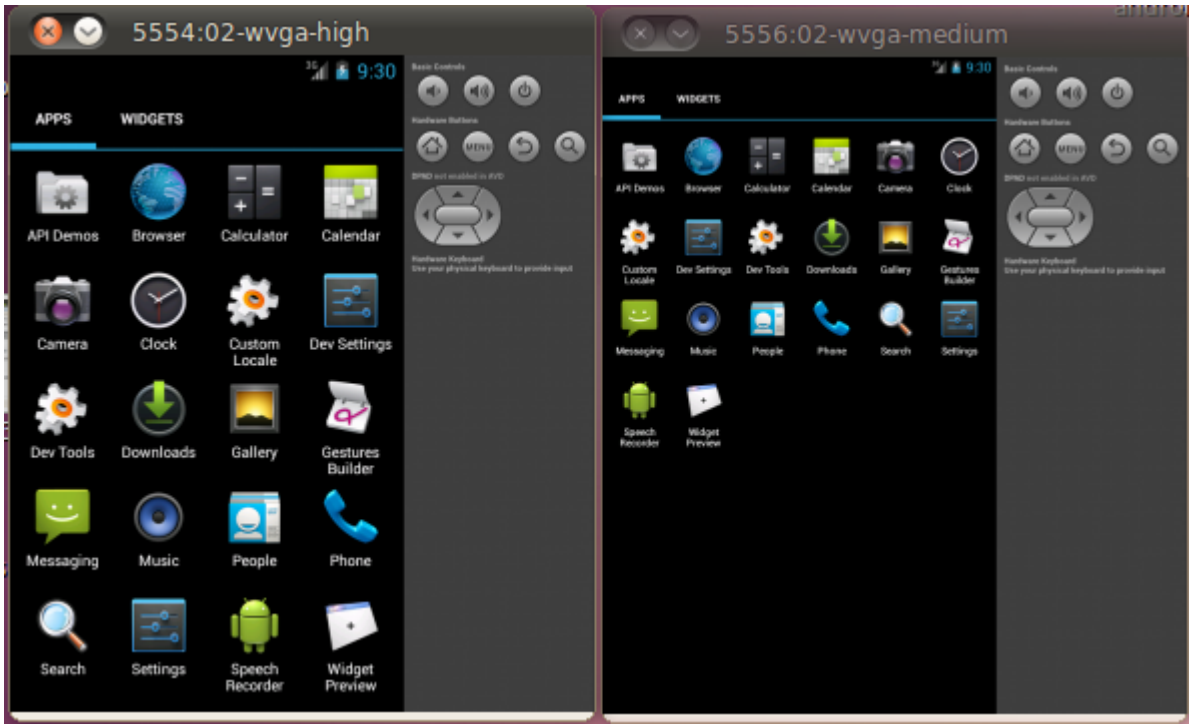
- **Generalized size and density**
  - Applications should be concerned only with generalized screen size and density groups
  - Each generalized size and density spans a range of actual screen sizes and densities
  - Screen-size buckets deprecated for Android >3.0

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

Resolution v. Generalized size

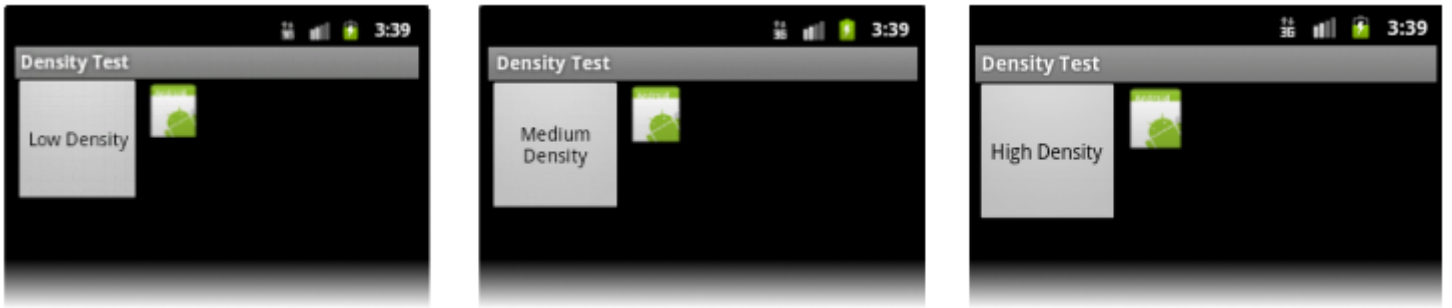
- WVGA (800 x 480)
- High-density (240 dpi)
- **Normal** generalized size

- WVGA (800 x 480)
- Medium-density (160 dpi)
- **Large** generalized size



Density independence

- Application preserves physical size of user interface elements when displayed on different densities
- UI element appears physically larger on a low density screen without "density independence"

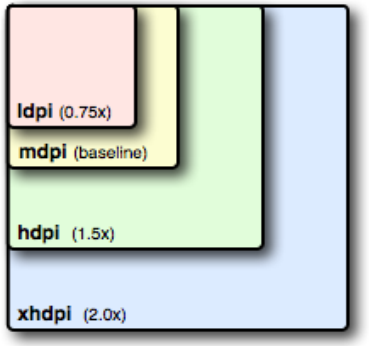


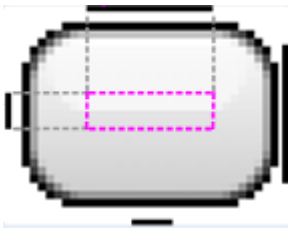
Why should I care?

- To provide optimized resources for application to run on different hardware
- Bitmap scaling can result in blurry or pixelated bitmaps. Use alternative bitmap resources for different densities instead
- To maximize user experience (UX) for all devices
- To make users believe application is designed for their devices - rather than simply stretched
- Typically, alternative layouts for different screen sizes and alternative images for different densities

Alternative drawables

- Almost every application should have alternative drawable resources for different screen densities
- Follow the 3:4:6:8 scaling ratio between the four generalized densities
- Nine-patch drawables
  - standard PNG image that includes an extra 1-pixel-wide border to define the stretchable and static areas of the image, or padding lines





Best practices

- Do not use hard-coded pixel values in your application code
- Use size and density-specific resources
- Use wrap_content, fill_parent, or the dp unit for layout dimensions
- Test Your Application on Multiple Screens

Android resources

Resources

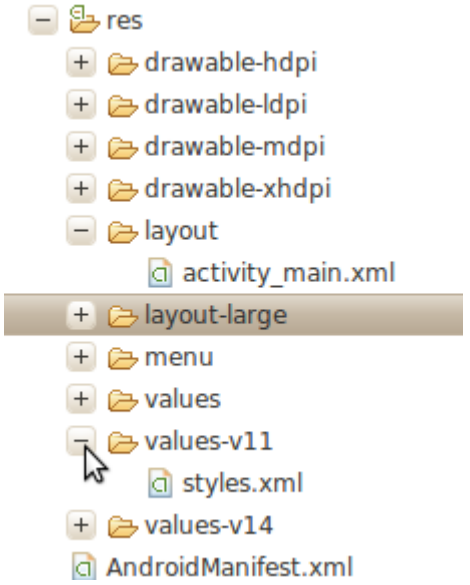
- Android supports the **externalization** of resources
  - strings, colors, images,

    themes, menus, layouts
  - system resources

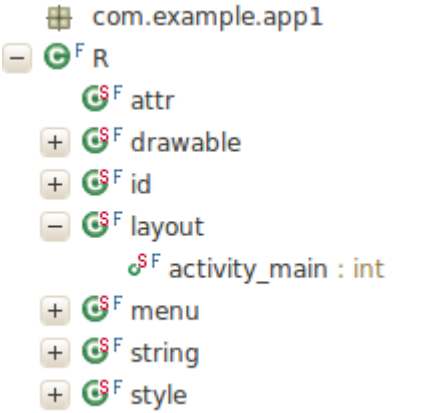
- Why?
  - easier to **maintain, update, and manage**
  - easier to define alternative resource values for internationalization and to support variations in hardware

res/ & R

- **res/**
  - Generated by ADT Wizard
  - Each resource type is stored in a different subfolder

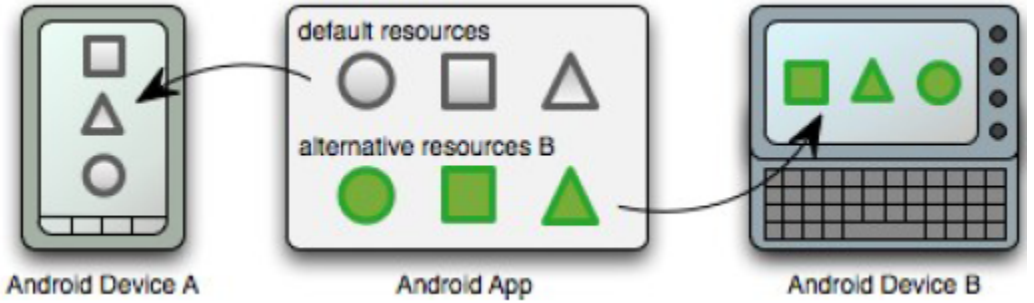


- **R.java**
  - Resources from /res are „indexed“ by ADT or aapt tool
  - Enables to reference resources in code



Default v. Alternative Resources

- For any type of resource, we can specify default and multiple alternative resources
- Default resources are used regardless of the device configuration when no alternative resources match the current configuration
- Alternative resources are designed for use with a specific configuration



How to specify Alternative Resources

- Create a new directory in res/

  res/<resources_type>-<config_qualifier>

- <config_qualifier> specifies a configuration for which these resources are to be used
- Save your alternative resources in new directory and name it as the default resource files

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

Configuration qualifiers

- Control how the system selects your alternative resources based on the characteristics of the current device screen

Mobile country code	mcc310, mcc310-mnc004
Language and region	en, fr, en-rUS, fr-rFR, fr-rCA
Smallest Screen Width **Available Screen Width** **Available Screen Height**	sw600dp, sw320dp w600dp, w320dp h720dp, h480dp * Android >3.0
Screen size	small, normal, large, xlarge
Orientation	land, port
Screen density	ldpi, mdpi, hdpi, nodpi
...	

Resource types

- **Simple values**
  - Strings
  - Colors
  - Dimensions
  - Styles
  - String or integer arrays

- Stored within XML files in the **res/values** folder

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="test_name">To Do List</string>

    <plurals name="androidPlural">
        <item quantity="one">One android</item>
        <item quantity="other">%d androids</item>
    </plurals>

    <color name="app_background">#FF0000FF</color>

    <dimen name="default_border">5px</dimen>

    <string-array name="string_array">
        <item>Item 1</item>
        <item>Item 2</item>
        <item>Item 3</item>
    </string-array>

    <array name="integer_array">
        <item>3</item>
        <item>2</item>
        <item>1</item>
    </array>
</resources>
```

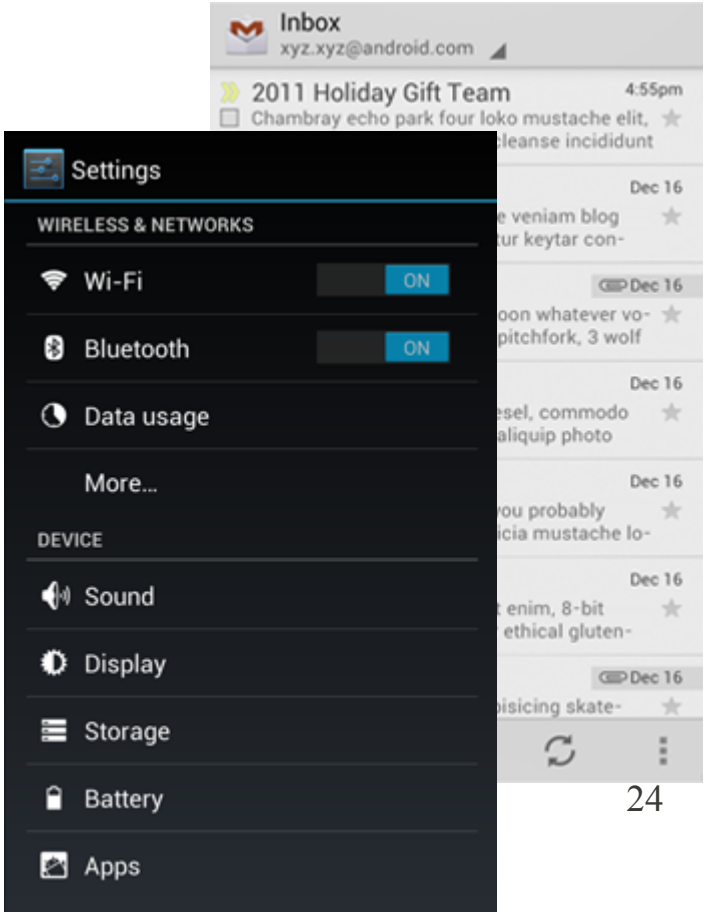
Resource types

- **Styles and Themes**
  - Let your applications maintain a consistent look and feel by enabling you to specify the attribute values used by Views
  - The most common use of is to store the colors and fonts for an application

```
<resources>

    <!--
        Base application theme, dependent on API level. This
        by AppBaseTheme fro res/values-vXX/styles.xml on new
    -->
    <style name="AppBaseTheme" parent="android:Theme.Light">
        <!--
            Theme customizations available in newer API level
            res/values-vXX/styles.xml, while customizations r
            backward-compatibility can go here.
        -->
    </style>

    <!-- Application theme. -->
    <style name="AppTheme" parent="AppBaseTheme">
        <!-- All customizations that are NOT specific to a pa

        <item name="android:textSize">14sp</item>
        <item name="android:textColor">#111</item>

    </style>

</resources>
```

Themes

- Themes are Android's mechanism for applying a consistent style to an app or activity
  - visual properties of the elements of user interface, such as color, height, padding and font size
- Three system themes
  - Holo Light
  - Holo Dark
  - Holo Light with dark action bars



Resource types

- **Layouts**
  - Construct screens in XML
  - Decouple layout from code
  - Optimized layouts
  - Layouts define the UI for any visual component, including Activities, Fragments, and Widgets

```
<RelativeLayout xmlns:android="http://schemas.androi
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/txtHi"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Accessing Resources

- **In Code** - pass the resource ID as a method parameter
  - `setContentView(`
    `R.layout.main_screen);`
  - `msgTextView = (TextView)`
    `findViewById(R.id.msg);`
    `msgTextView.setText(`
    `R.string.hello_message;`
- **In XML** - define values for some XML attributes and elements using a reference to an existing resource

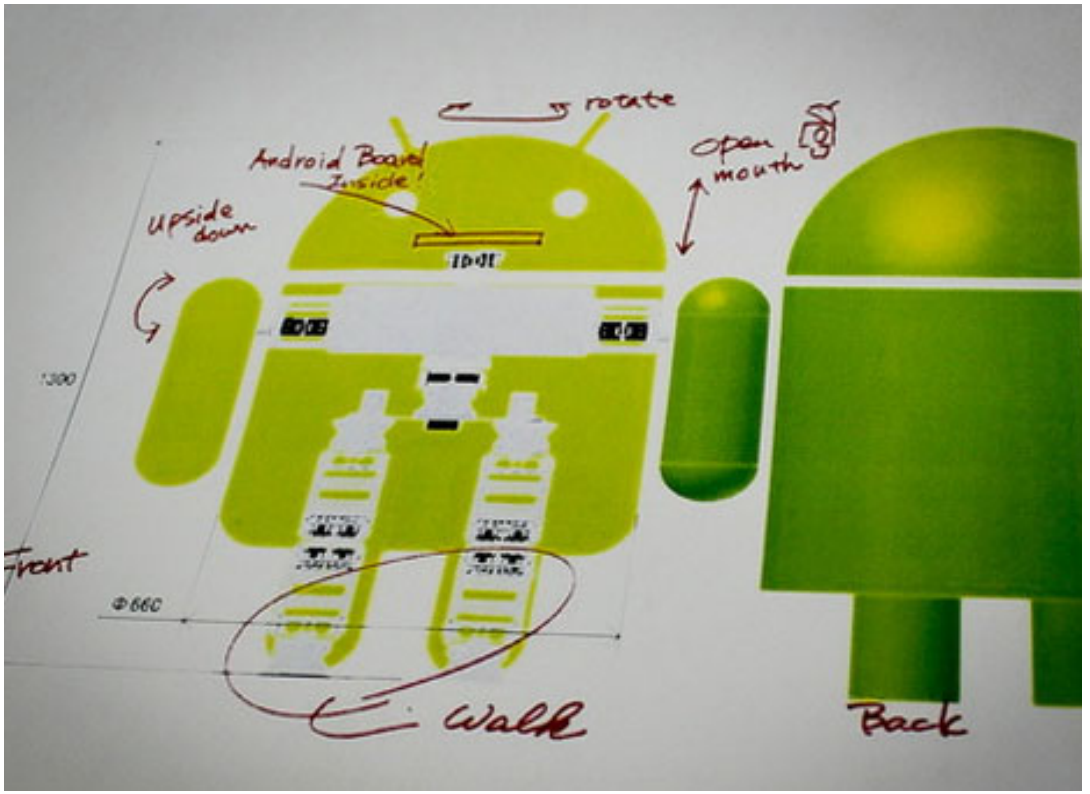
```
<Button
  android:layout_width=
  "fill_parent"
  android:layout_height=
  "wrap_content"
  android:text=
  "@string/submit"/>
```

Localization

- Application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your application will be used
- Use Android resource framework to separate localized aspects of your application from the code
  - create Default Resources
  - put language-specific resources to

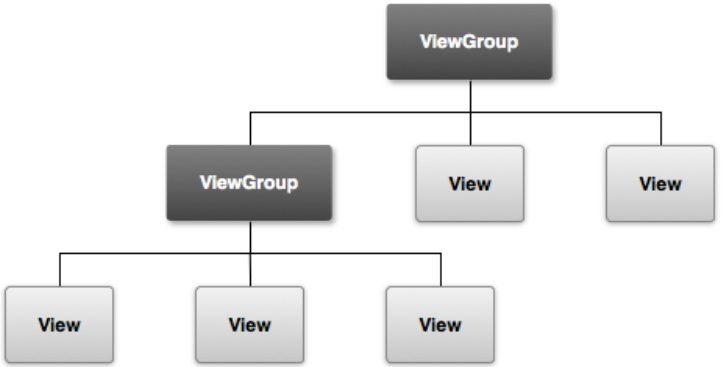
    res/values-<language-code>/strings.xml

Building User Interfaces



View

- **View**
  - base class for all visual interface elements controls, widgets
  - an object that draws something on the screen that the user can interact with
- **View Group**
  - extends View class
  - an object that holds other Views



View group is an invisible container that organizes child views, while the child views draw some part of the UI

Layout

- Defines the visual structure for a user interface, such as the UI for an activity or app widget.
- Options to declare a Layout
  - Declare UI elements in XML (preferred)
  - Instantiate layout elements at runtime (programmatically in Java code)
  - both options

Layout

- Advantages of declaring UI in XML
  - Separation of the presentation from the code that controls its behavior
    - modify UI without having to modify source code
    - create XML layouts for different screen orientations, different device screen sizes, and different languages
  - Easier to visualize the structure of your UI
    - Easier to design/debug UI
    - Visualizer tool (ADT)

Attributes

- View and ViewGroup element attributes
  - View specific
  - Common to all View elements
  - Layout parameters, describe layout orientations of the View element

- **ID attribute** (android:id="@+id/element")
  - Any View object may have an integer ID associated with it, to uniquely identify the View within the tree

Layout file structure

- Hierarchical tree structure of ViewGroups and Views
- Contain one root element, either
  - ViewGroup element (e.g., LinearLayout)
  - View element (e.g. Button)
- Layout resource is loaded in Activity

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

Layout File Structure

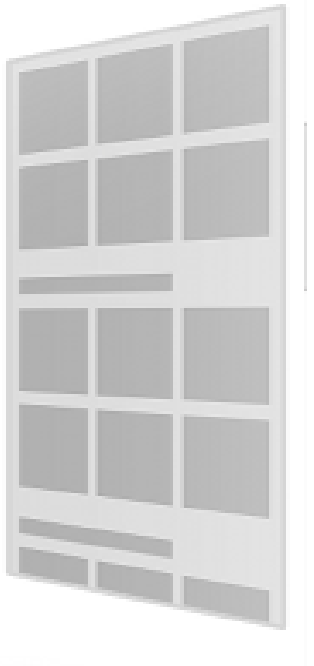
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        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>
```

ViewGroup

View

Layout Types

- **LinearLayout**
  - Aligns all children in a single direction - vertically or horizontally
  - All children are stacked one after the other
  - Layout weight (android:layout_weight) assigns an "importance" value to a view in terms of how much space is should occupy on the screen

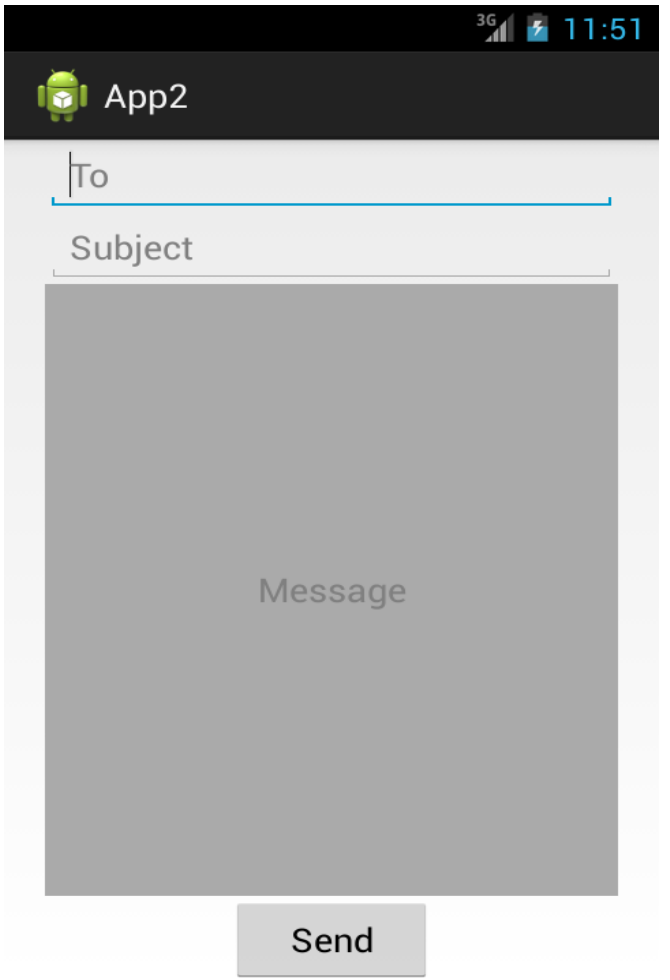


FrameLayout

- simplest type of layout object
- a blank space on your screen that you can later fill with a single object
- child elements are pinned to the top left corner

LinearLayout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/ap
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:paddingLeft="20dp"
    android:paddingRight="20dp" >

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/toHint" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subjectHint" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="@android:color/darker_gray"
        android:gravity="center"
        android:hint="@string/messageHint" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/send" />
</LinearLayout>
```



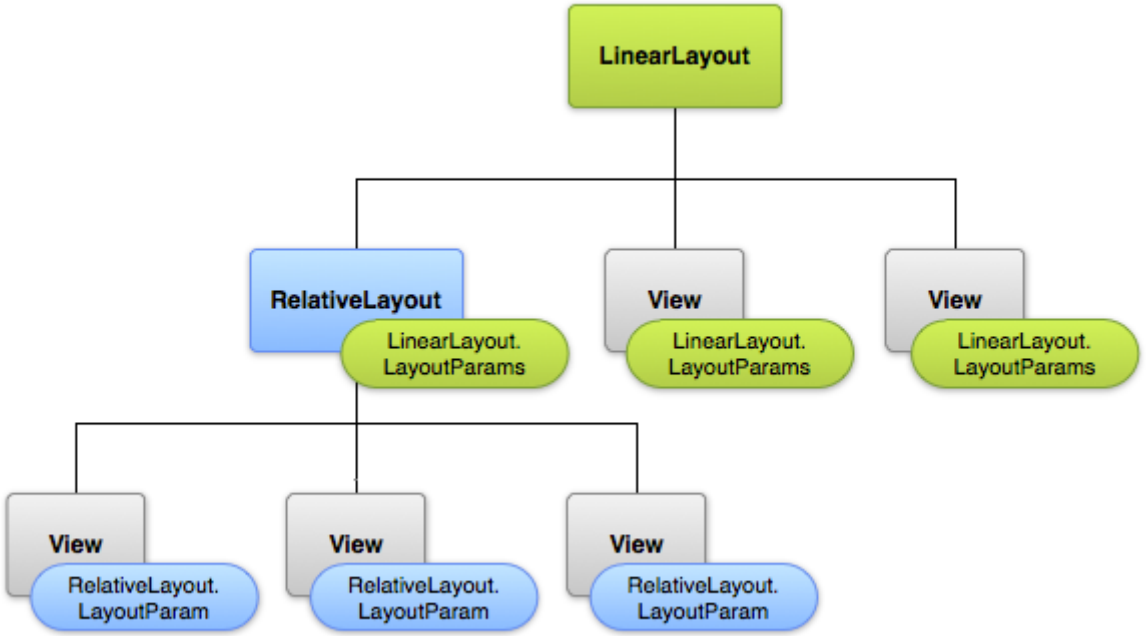
Layout types

- **RelativeLayout**
  - view group that displays child views in relative positions
  - can eliminate nested view groups and keep your layout hierarchy flat, that improves performance
  - specify the location of child objects relative to each other (child A to the left of child B)

    or to the parent

Layout types

- **Layout Parameters** (layout_something)
  - parent view group defines layout parameters for each child view (including the child view group)
  - child element must define LayoutParams that are appropriate for its parent

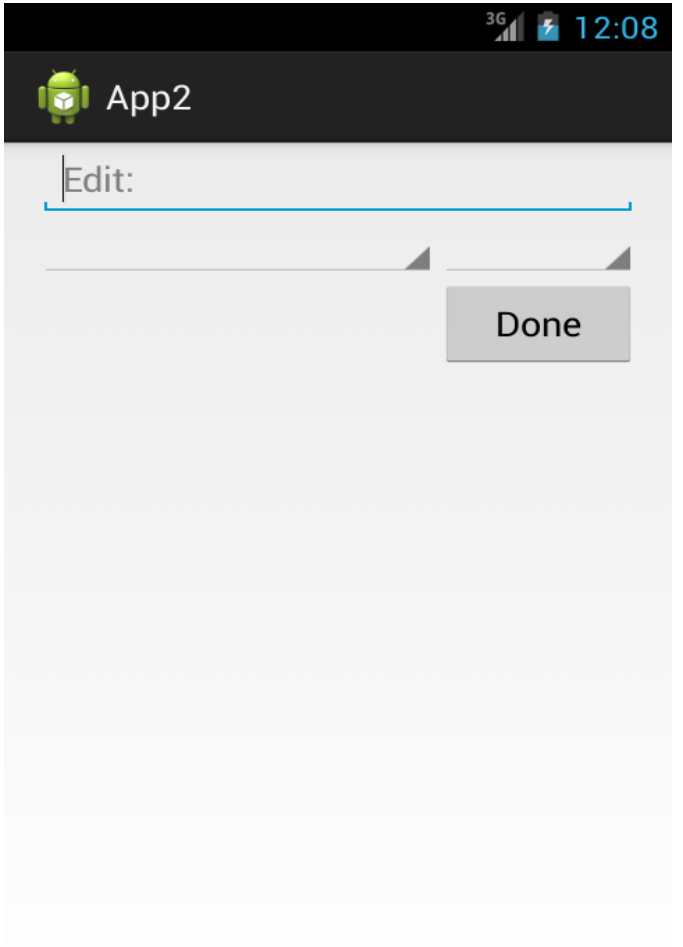


Layout types

- **RelativeLayout.LayoutParams**
  - android:layout_alignParentTop - top edge of this view match the top edge of the parent
  - android:layout_centerVertical - centers this child vertically within its parent.
  - android:layout_below - put top edge of this view below the view specified
  - android:layout_toRightOf - put left edge of this view to the right of the view specified

RelativeLayout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.andr
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



Optimizing Layouts

- inflating layouts is an expensive process
- keep your layouts as simple as possible
- good practice to restrict nesting to 10 levels
- use Lint tool to analyze your layouts
- use “Merge” and “Include” tags
  - when a layout containing a merge tag is added to another layout, its child Views are added directly to the new parent

Basic widget toolbox

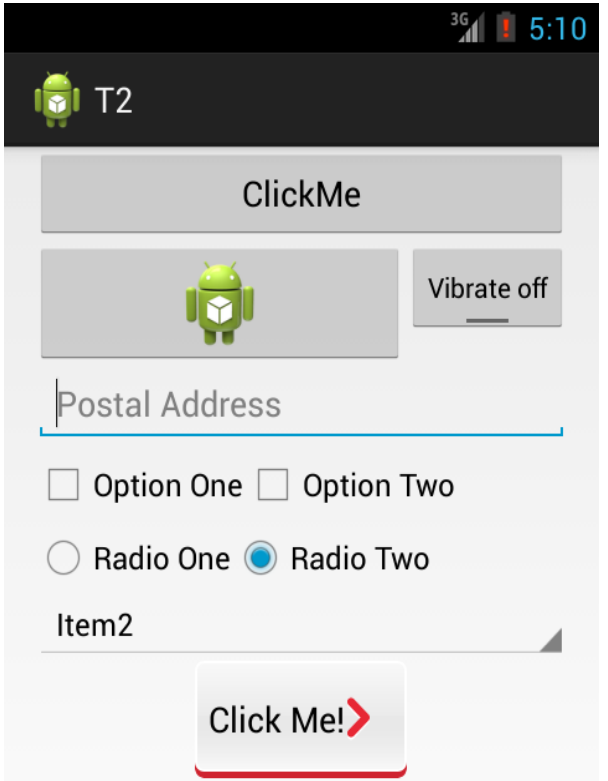
Button, ImageButton	A push-button that can be pressed, or clicked
EditText, AutoCompleteTextView	An editable text field. AutoCompleteTextView provides auto-complete suggestions
CheckBox	An on/off switch that can be toggled by the user.
RadioGroup RadioButton	Similar to checkboxes, except that only one option can be selected in the group.
ToggleButton	An on/off button with a light indicator.
Spinner	A drop-down list that allows users to select one value from a set.
DatePicker, TimePicker	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture.

Basic widget toolbox. Buttons

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text" />
```

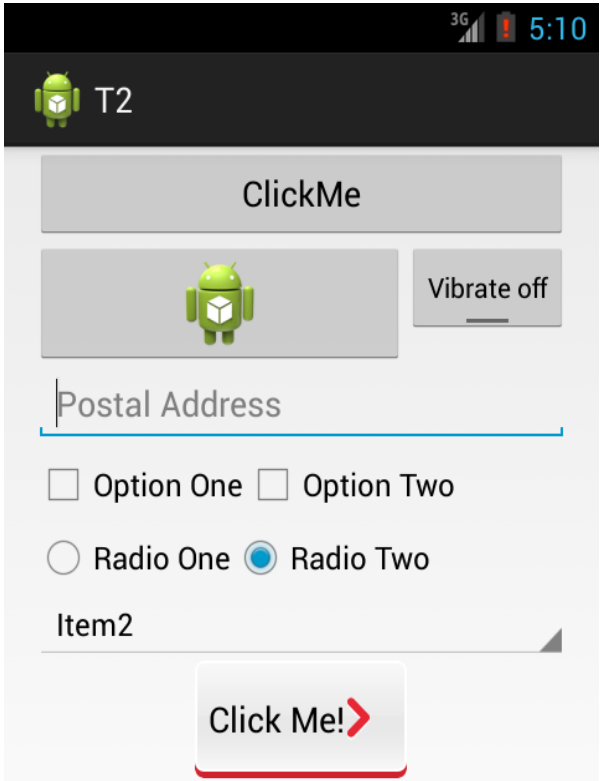
```
<ImageButton
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:contentDescription="@string/description"
    android:src="@drawable/ic_launcher" >
</ImageButton>
```

```
<ToggleButton
    android:id="@+id/togglebutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="Vibrate off"
    android:textOn="Vibrate on" />
```



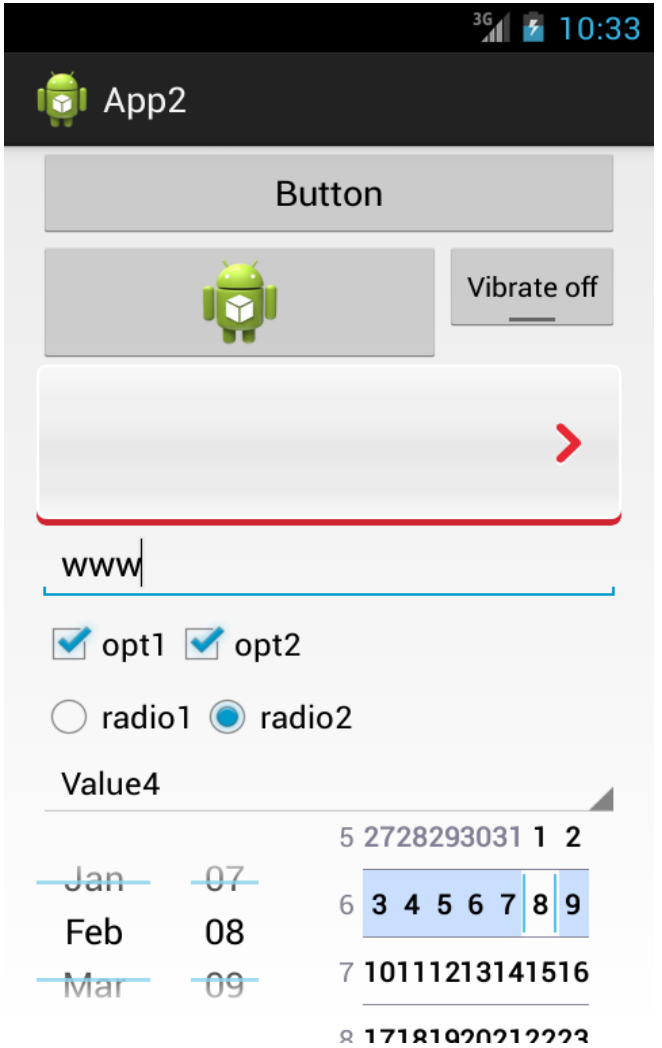
Basic widget toolbox. Inputs

```
<EditText
    android:id="@+id/postal_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/postal_address_hint"
    android:inputType="textPostalAddress" />

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:paddingTop="5dp" >

    <CheckBox
        android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat" />

    <CheckBox
        android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese" />
</LinearLayout>
```



Basic widget toolbox. Inputs

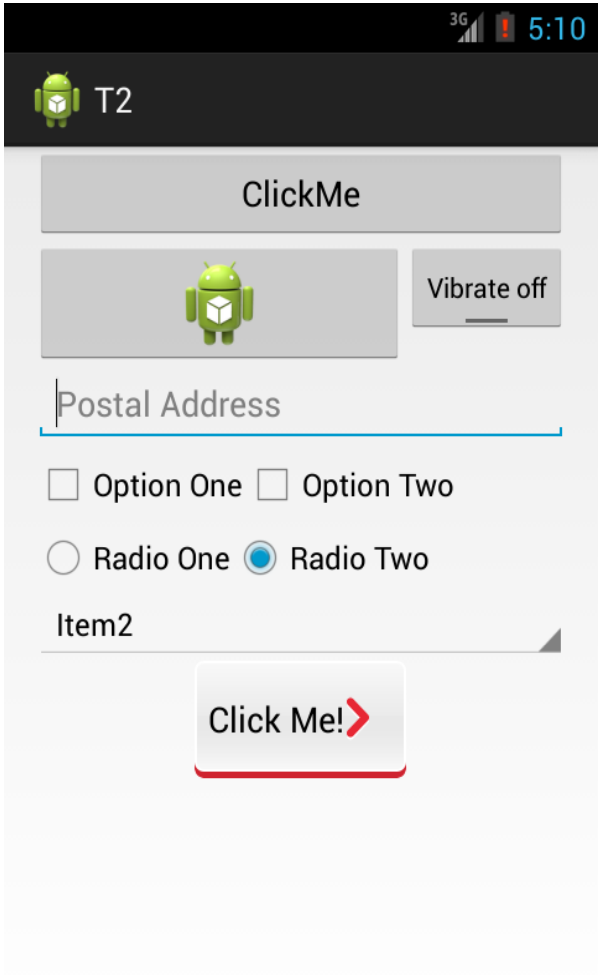
```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:paddingTop="5dp" >

    <RadioButton
        android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates" />

    <RadioButton
        android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas" />
</RadioGroup>
```



Basic widget toolbox. Custom button

```
<Button
    android:id="@+id/lng_button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:background="@drawable/button_custom"
    android:padding="@null"
    android:text="Click Me!"/>
```

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:drawable="@drawable/lng_btn_dwn" android:state_pressed="true"/>
    <item android:drawable="@drawable/lng_btn_dwn" android:state_focused="true"/>
    <item android:drawable="@drawable/lng_btn_up"/>

</selector>
```

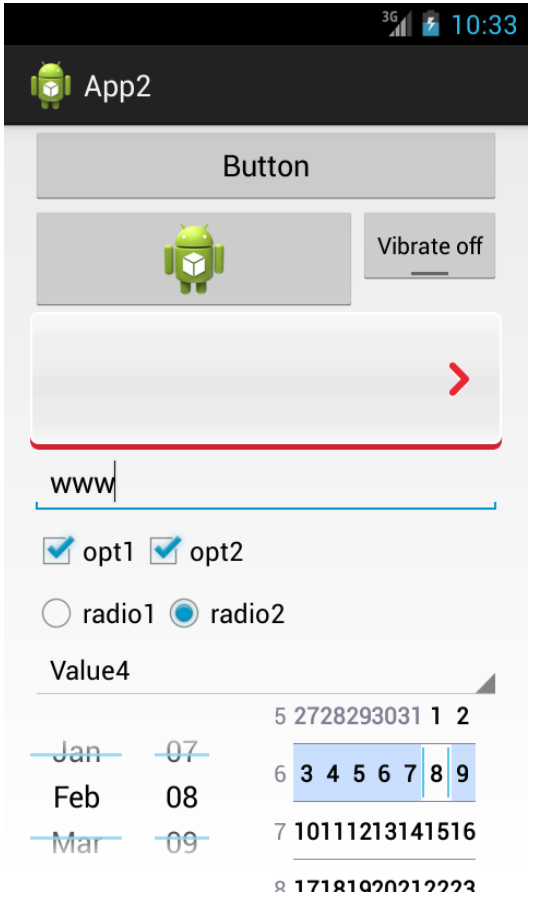


Basic widget toolbox. Spinner

```
<Spinner
    android:id="@+id/spinner1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

<DatePicker
    android:id="@+id/datePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

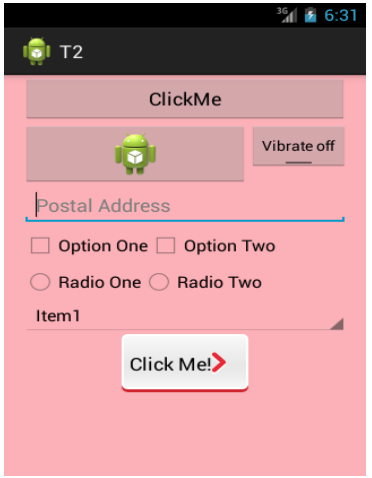
```
<string-array name="spinnerValues">
    <item>Value1</item>
    <item>Value2</item>
    <item>Value3</item>
    <item>Value4</item>
</string-array>
```

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

    // find spinner component
    final Spinner spinner = (Spinner) findViewById(R.id.spinner1);

    // prepare array adapter
    final ArrayAdapter<CharSequence> adapter =
            ArrayAdapter.createFromResource(this,
                    R.array.spinnerValues, android.R.layout.simple_spinner_item);

    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    spinner.setAdapter(adapter);
}
```

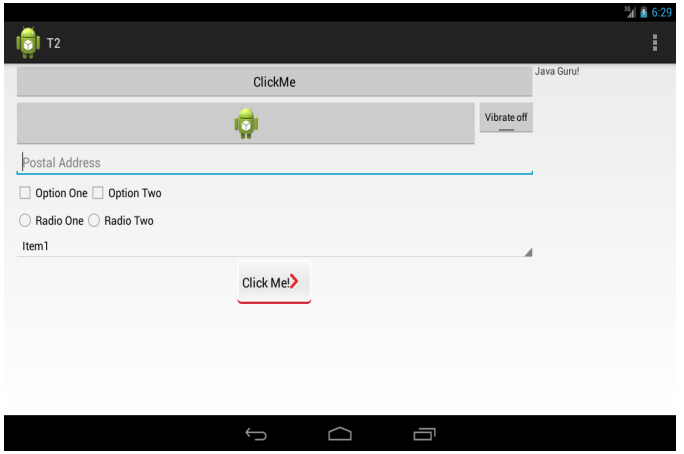


Practical task

- Prepare 3 different AVD configurations
  - Phone - 320x480 (HVGA), 3.2", 160dpi
  - Tablet - 1024x600 (WSVGA), 7", 160dpi
  - Nexus - 480x800 (WVGA), 4.0", 240dpi



- Create application with screen layout consisting of basic widgets (like in slides). On tablet additional section on right with banner "Java Guru" should appear. NB! Respect layout reuse.



Practical task

- Add custom button with provided background images (up and down states). NB! Consider density and image correct scaling (use draw9patch took if needed).
- Fill Spinner with some values using string array resource and localize application for LV.
- Draw layout that looks like provided.