Trường Đại học Bách Khoa Hà Nội
Hanoi University of Science and Technology

Android GUI Components

Chapter 3

Do Trong Tuan – Pham Van Tien

Content

App fundamentals

- Android applications are composed of one or more application components (activities, services, content providers, and broadcast receivers)
- Each component performs a different role in the overall application behavior, and each one can be activated individually (even by other applications)
- The manifest file must declare all components in the application and should also declare all application requirements, such as the minimum version of Android required and any hardware configurations required
- Non-code application resources (images, strings, layout files, etc.) should include alternatives for different device configurations (such as different strings for different languages and different layouts for different screen sizes)

Android Application Building Blocks

- App fundamental
- Activities
- Intents
- Services
- Content Providers
- Broadcast Receivers
- Notifications

Activities

- Typically correspond to one UI screen
- Activities can:
  - Be faceless
  - Be in a floating window
  - Return a value

✓ *Every screen in an application will be an extension of the Activity class.*

✓ *An activity as being analogous to a window or dialog in a desktop environment.*

Intents

- Think of Intents as a verb and object; a description of what you want done
  - E.g. VIEW, CALL, PLAY etc..
- System matches Intent with Activity that can best provide the service
- Activities and IntentReceivers describe what Intents they can service

✓ *A simple message passing framework. Using intents you can broadcast messages system-wide or to a target Activity or Service.*

Services

- ❖ Faceless components that run in the background
  - E.g. music player, network download etc…

✓ *Services are designed to keep running independent of any activity.*

Content Providers

- Enable sharing of data across applications
  - E.g. address book, photo gallery
- Provide uniform APIs for:
  - querying
  - delete, update and insert.
- Provide a level of abstraction for any data stored on the device that is accessible by multiple applications

Broadcast Receivers

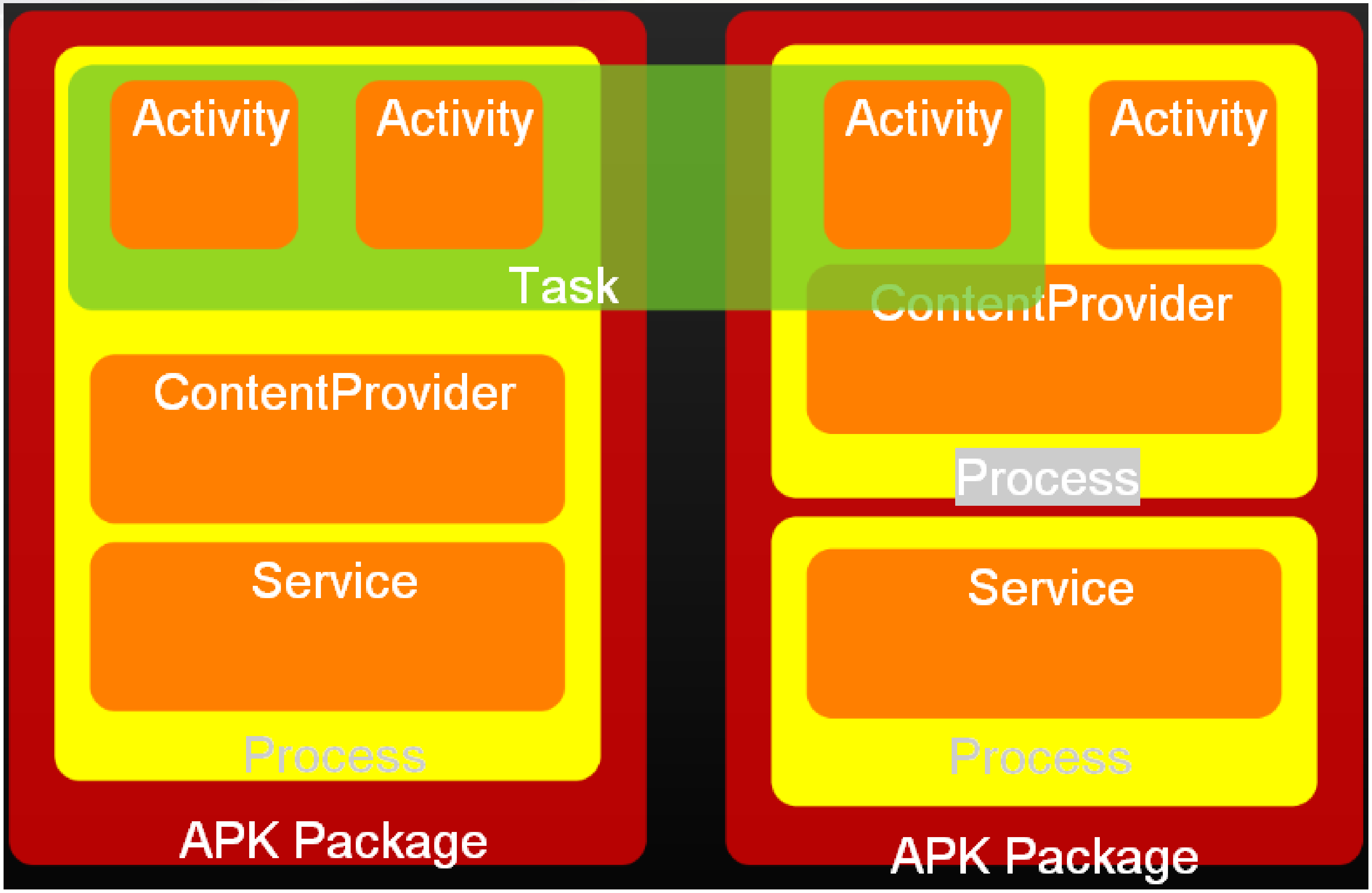
- Components that respond to broadcast ‘Intents’ (*Intent broadcast consumers*).
- By registering a broadcast receiver an application can listen for broadcast Intents that match specific filter criteria.
- Way to respond to external notification or alarms

Notifications

- User notification framework.
- To signal users without interrupting their current activity.

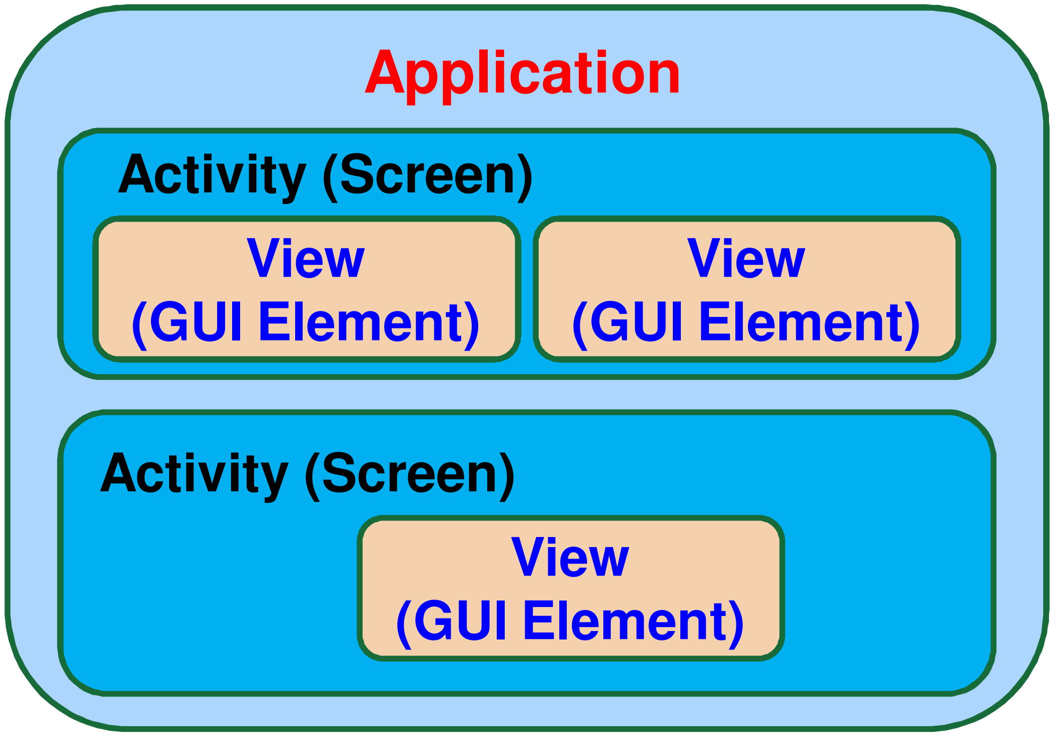
✓ *For instance an incoming call can alert users with flashing lights, making sounds, or showing a dialog*

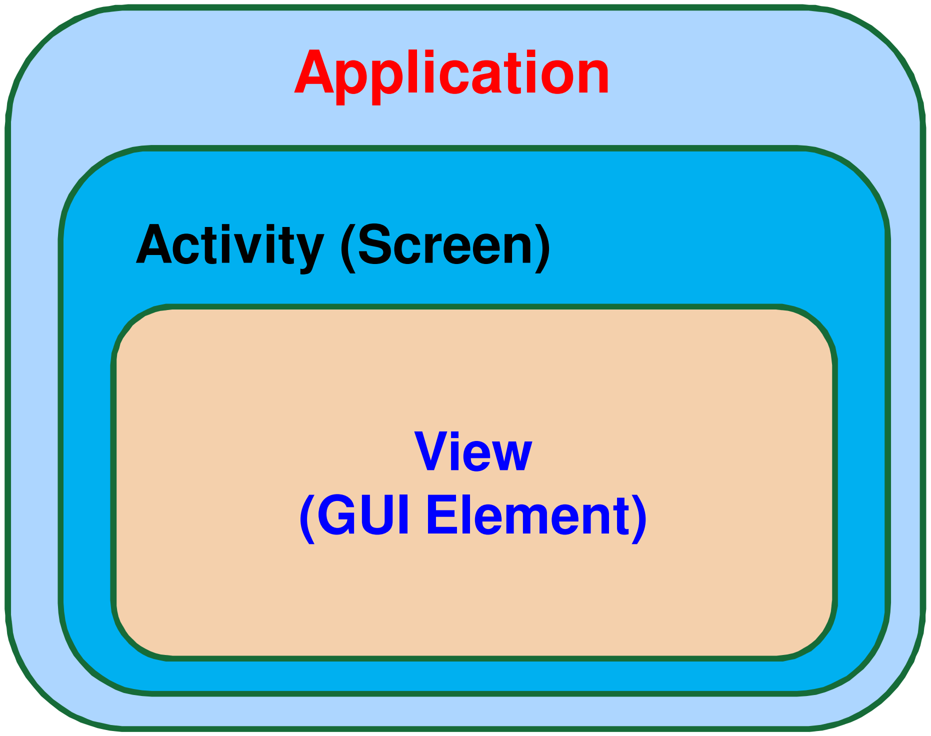
Program/Task & Activities



A Program/Task is a collection of Activities

Overview of GUI Components





An application with two screens:
(e.g: Main screen & Settings)
Each screen is composed of several GUI elements

An application with one Activity with one View

Overview of GUI Components

❖ **Views**:

– Single widgets or controls

– How the user interacts with your application

❖ **ViewGroups**:

– One or more views combined together

– Two uses:

- [Layouts](): Invisible, control the flow of other widgets
- [Advanced widgets](): Visible, implement complex controls

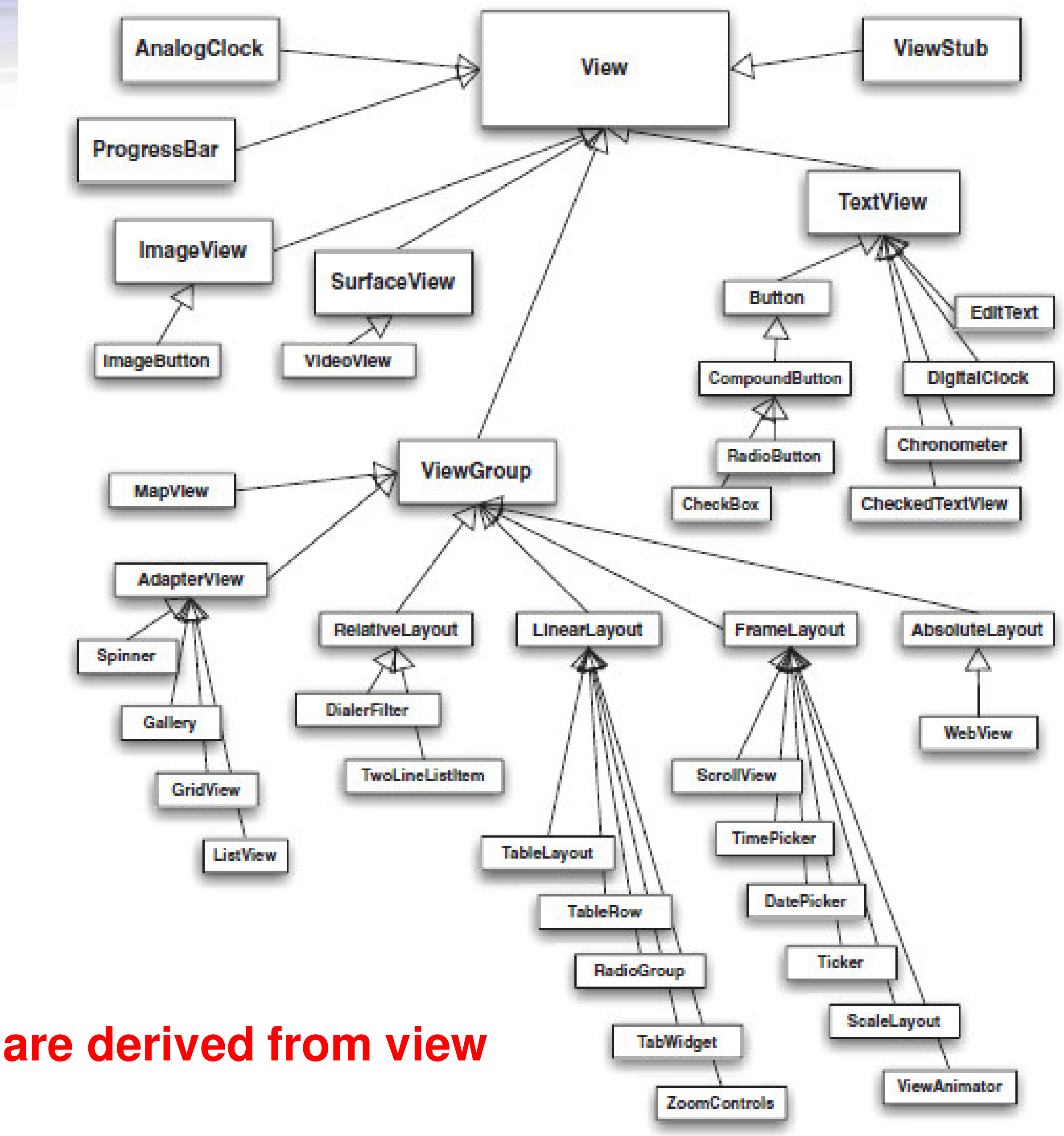
Making the elements of your GUI

VIEWS AND VIEWGROUPS

View

- An Android User Interface is composed of hierarchies of objects called Views.
- A View is a drawable object used as an element in your UI layout, such as a button, image etc…
- The User Interface of an Activities is build with widgets classes which inherent from "android.view.View".
- The layout of the views is managed by "android.view.ViewGroups".

View

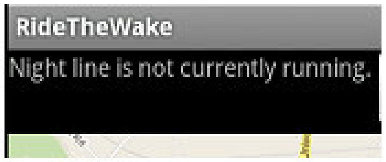


All screens are derived from view

View

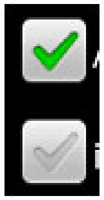
Some simple view items

- TextView

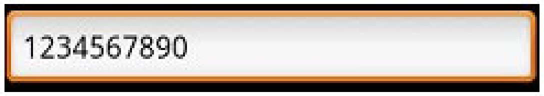


Night line is not currently running.

- CheckBox:

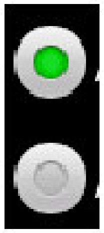


- EditText

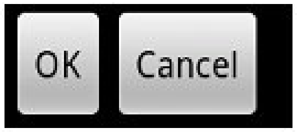


Can also be used as a password field

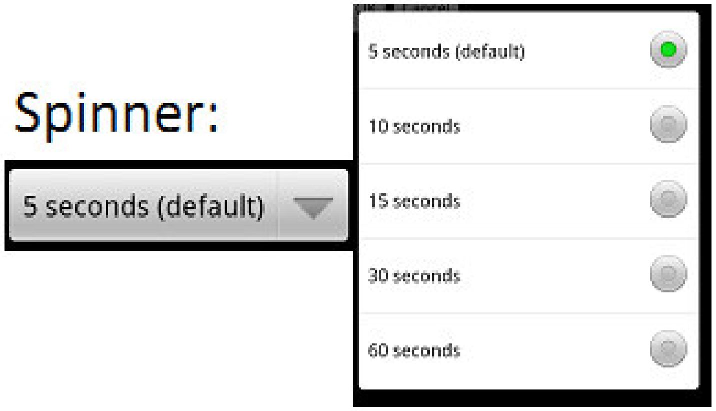
- RadioButton:



- Button:

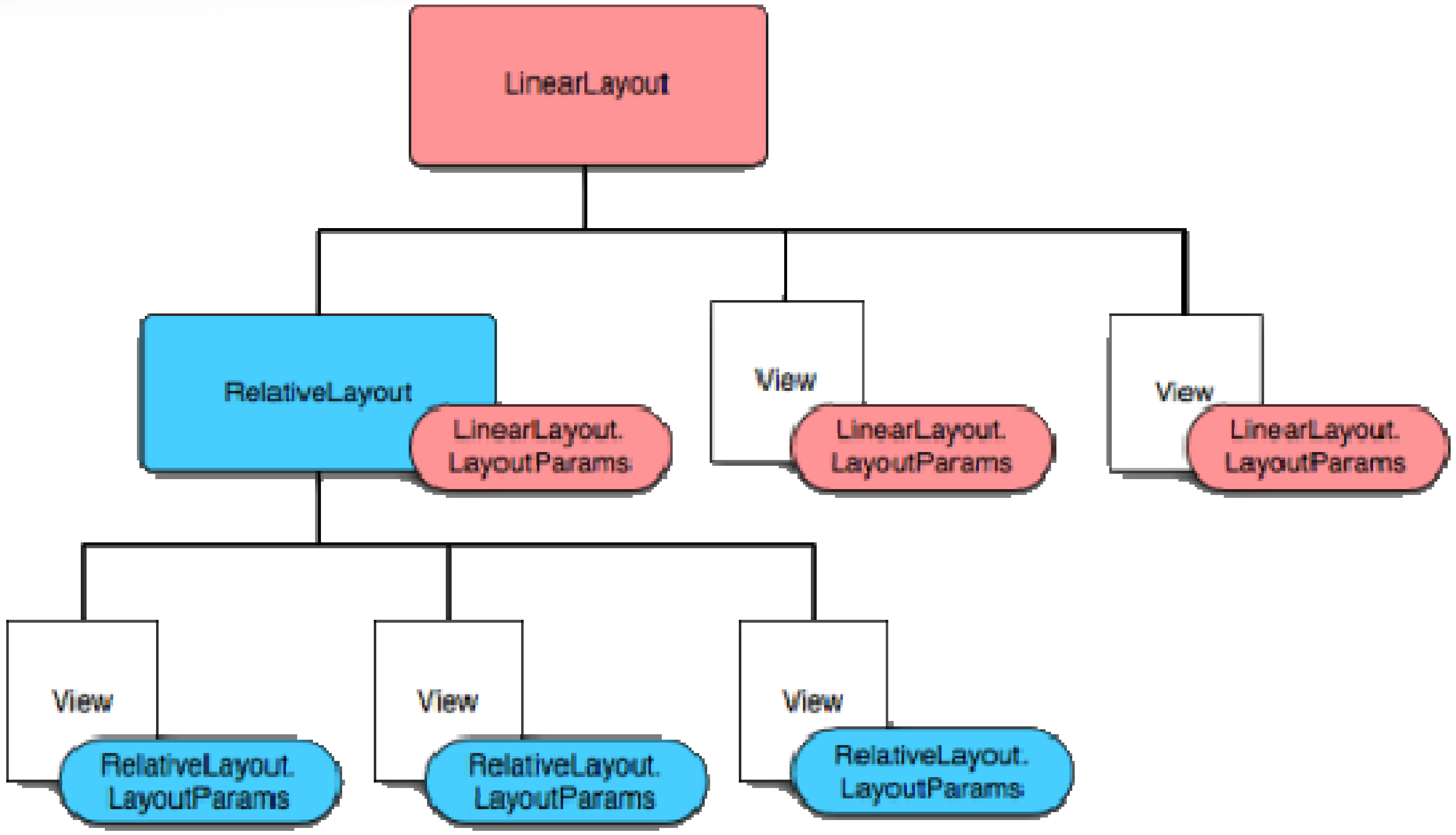


- Spinner:



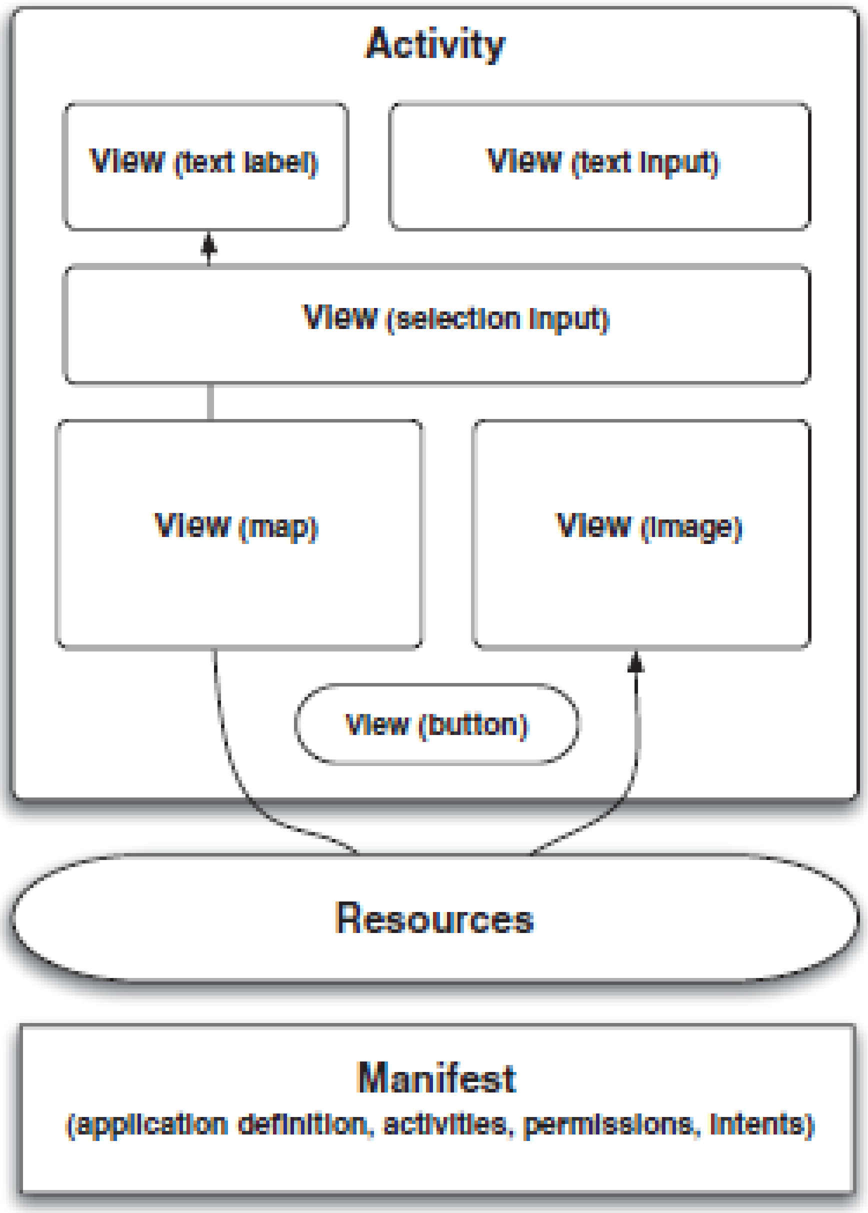
✓ a View ~ an object that knows how to draw itself on the screen

View


LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParams
View
RelativeLayout. LayoutParams
View
RelativeLayout. LayoutParams

All layouts are hierarchical

View

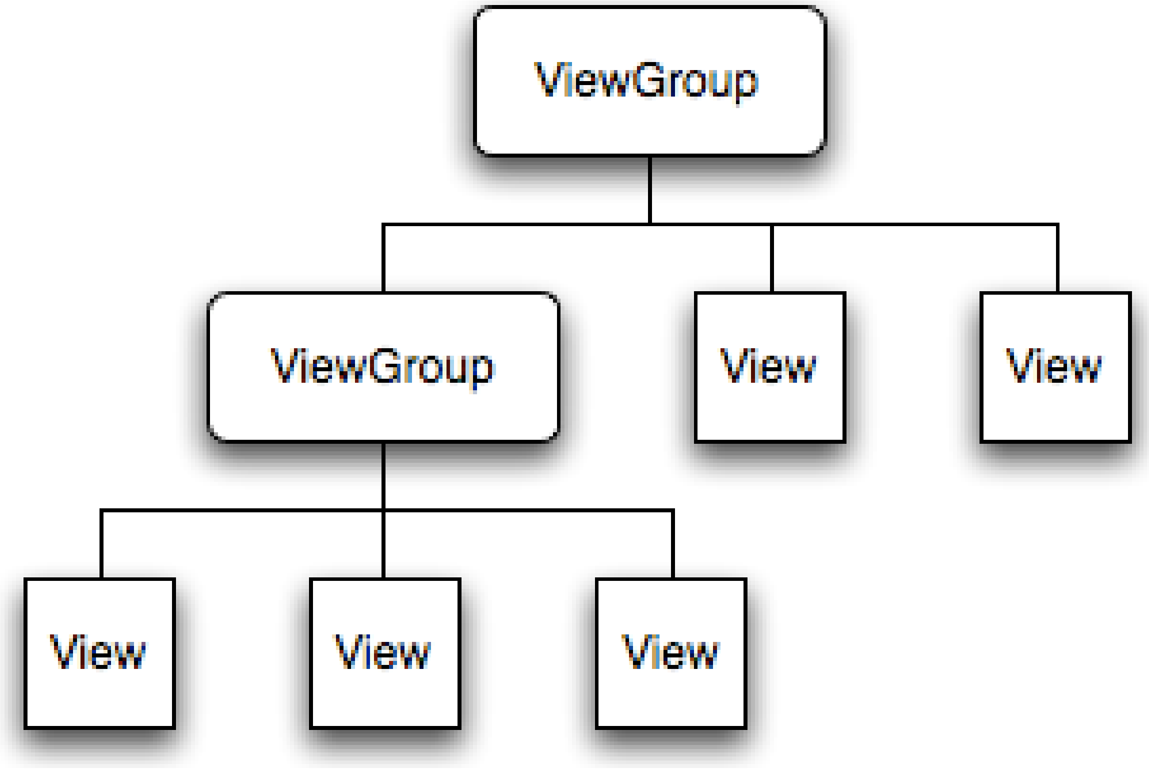


One class per activity, and screen, which may be done as xml file

Views are tied to activities (screens)

View Group

- Create more complex interfaces with Multiple Views but, also ViewGroups (in a XML layout file)



Views and ViewGroups

- An Activity can contain views and ViewGroups.
- android.view.View.* = base class for all Views.
  - example sub-classes include: TextView, ImageView, etc.
- android.view.ViewGroup = Layout for views it contains, subclasses include
  - **android.widget.LinearLayout**
  - android.widget.AbsoluteLayout
  - android.widget.TableLayout
  - android.widget.RelativeLayout
  - android.widget.FrameLayout
  - android.widget.ScrollLayout

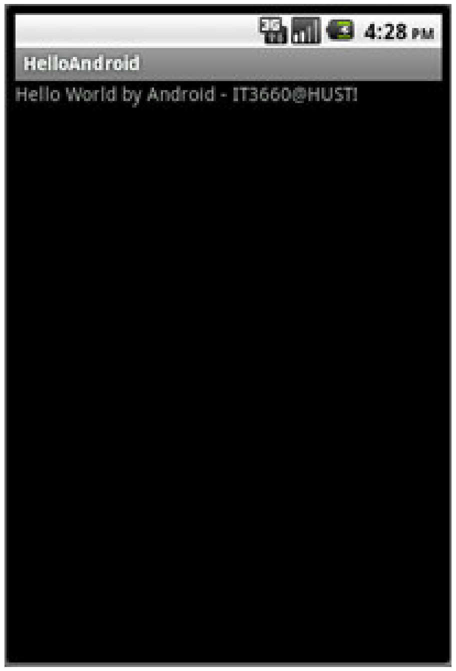
ViewGroups - Layouts

- Controls how Views are laid out
  - LinearLayout : single row or column
  - RelativeLayout : relative to other Views
  - TableLayout : rows and columns
  - FrameLayout : each child a layer
  - AbsoluteLayout : <x,y> coordinates

ViewGroups - LinearLayout

- Arranges by single column or row.
- Child views can be arranged vertically or horizontally.

```
<?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" >
 <Text View
      android:layout_width=“fill_parent”
     android:layout_height=“wrap_content”
     android:text=“@string/hello”/>
</LinearLayout>
```



Hello World by Android - IT3660@HUST!

Linear Layout Example

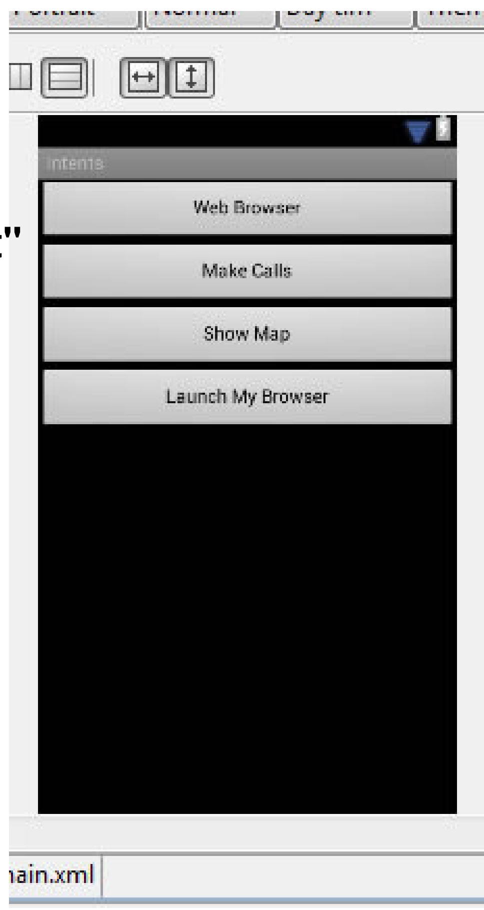
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent“
      android:orientation="vertical" >

     <Button android:id="@+id/btn_webbrowser" android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Web Browser“
      android:onClick="onClickWebBrowser" />

     <Button android:id="@+id/btn_makecalls" android:layout_width="fill_parent"
     android:layout_height="wrap_content“
      android:text="Make Calls"
     android:onClick="onClickMakeCalls" />

     <Button android:id="@+id/btn_showMap" android:layout_width="fill_parent"
     android:layout_height="wrap_content“
      android:text="Show Map"
     android:onClick="onClickShowMap" />

     <Button android:id="@+id/btn_launchMyBrowser"
     android:layout_width="fill_parent" android:layout_height="wrap_content"
     android:text="Launch My Browser" android:onClick="onClickLaunchMyBrowser" />
</LinearLayout>
```



View or ViewGroup Summary

Attribute	Description
layout_width	specifies width of View or ViewGroup
layout_height	specifies height
layout_marginTop	extra space on top
layout_marginBottom	extra space on bottom side
layout_marginLeft	extra space on left side
layout_marginRight	extra space on right side
layout_gravity	how child views are positioned
layout_weight	how much extra space in layout should be allocated to View (only when in LinearLayout or TableView)
layout_x	x-coordinate
layout_y	y-coordinate

Activity

- Presentation Layer of an Android application, *e.g. a screen which the user sees.*
- An Android application can have several activities and it can be switched between them during runtime of the application.

✓ *An initial Activity Class would be generated when you create an Android Project*

✓ *It will be setup to start at launch time*

✓ *Many Activity classes could be created in a project (with interfaces specified either by layout.xml files or generated in code).*

Activity

- A single, focused thing that the user can do
- Takes care of creating a window for user
- Presentation to the user
  - Ffull-screen windows
  - Floating windows
  - Embedding inside of another activity
- Lifecycle
  - void onCreate(Bundle *savedInstanceState*)
  - void onStart()
  - void onRestart()
  - void onResume()
  - void onPause()
  - void onStop()
  - void onDestroy()

Activity

- Within an application,there can be multiple Activities (screens)
- Activities are maintained in a stack

Activity States

✓**Active**

: Foreground – receiving input

✓**Paused**

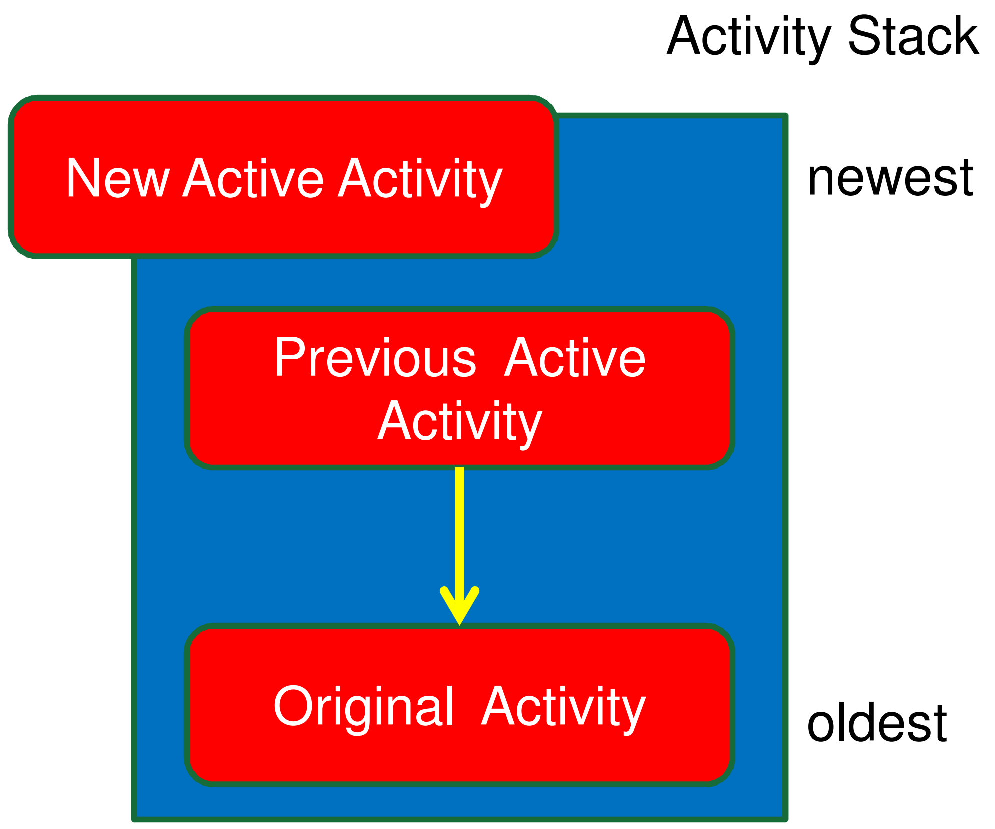
: Visible but obscured

✓**Stopped**

:No longer visible, still in memory

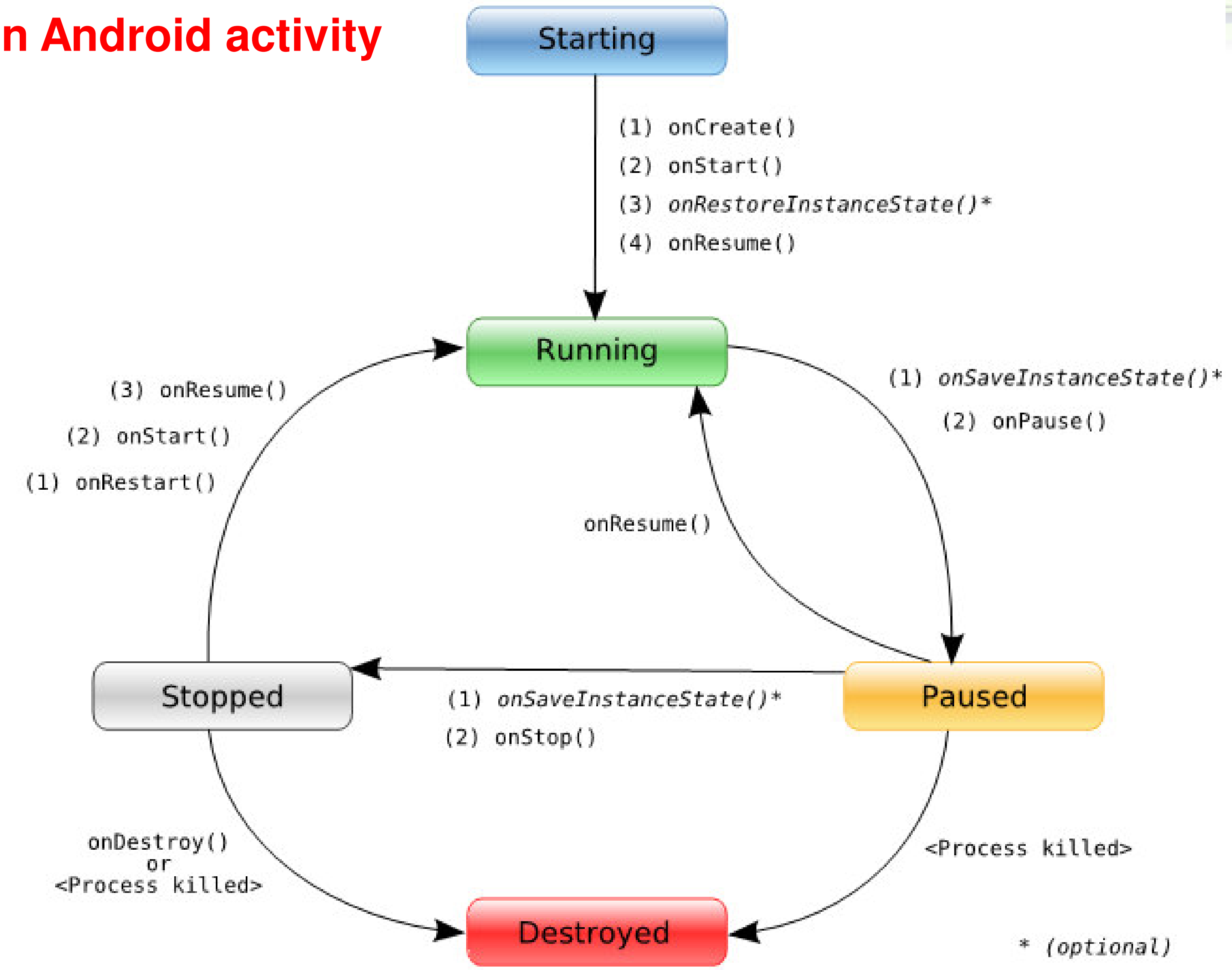
✓**Inactive**

: Not visible, not in memory (terminated)



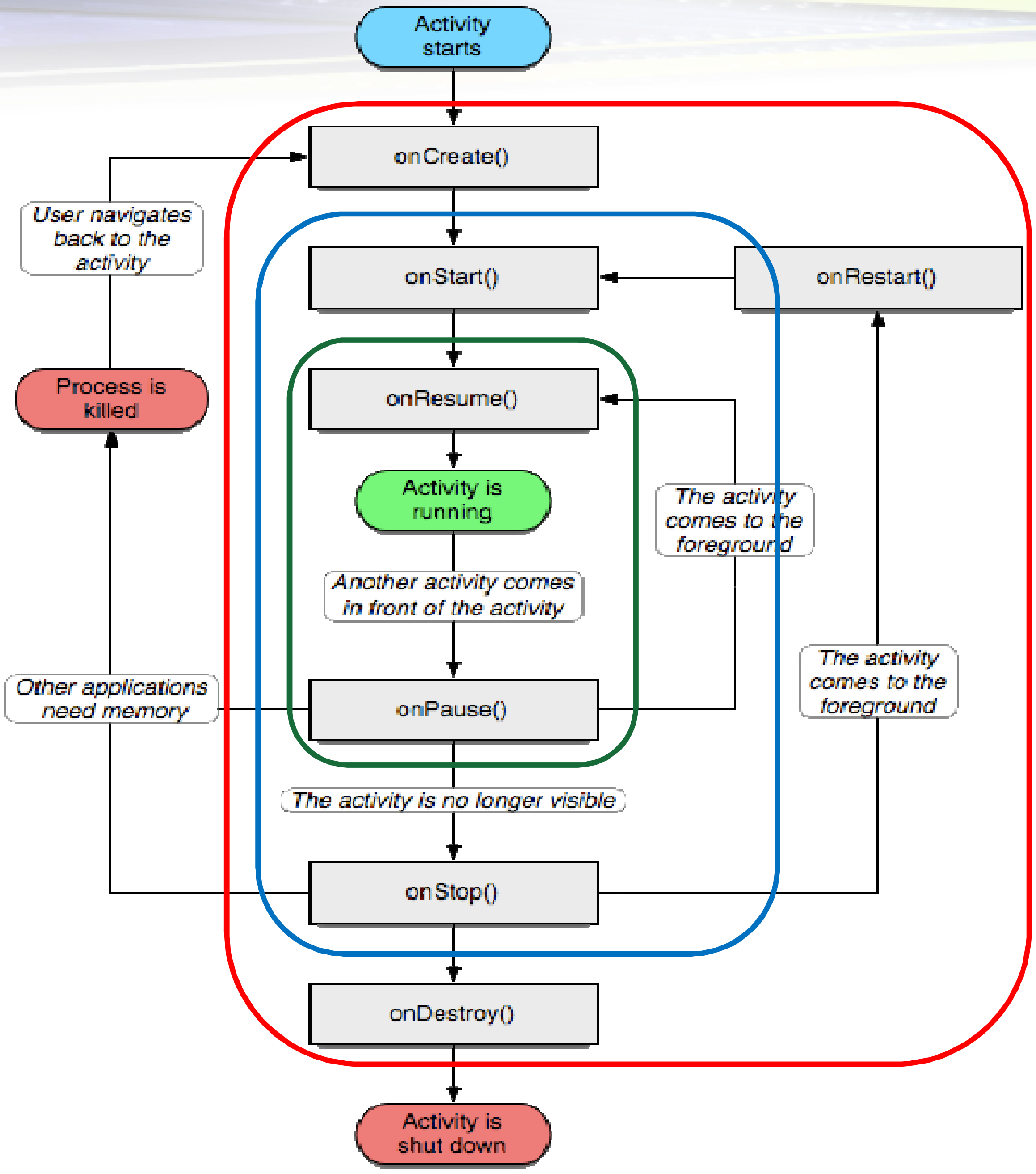
Activity

Life cycle of an Android activity

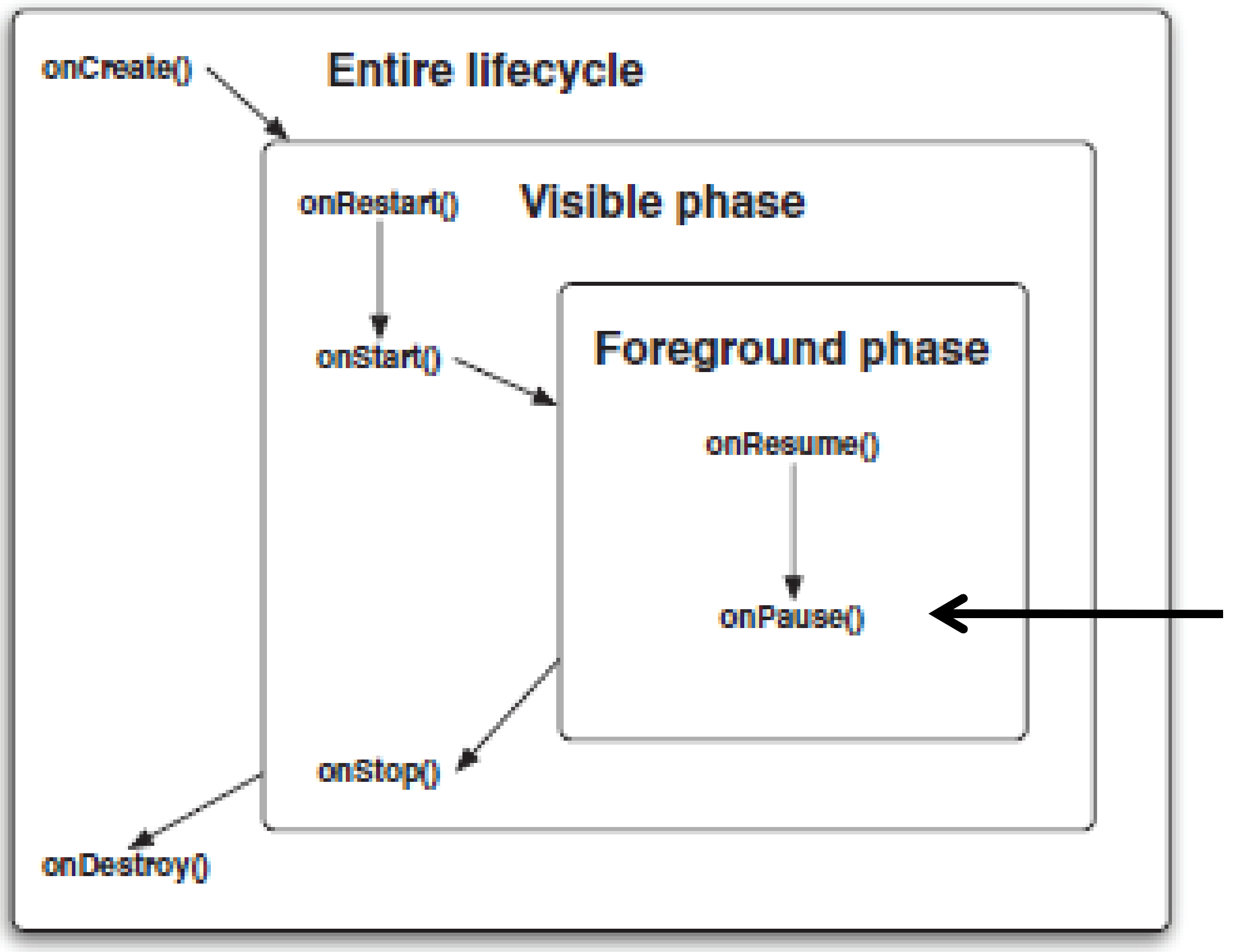


As an Activity moves through its possible different states, functions are automatically called on the Activity, triggering different parts of our code.

Activity Life Cycle


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Activity Life Cycle



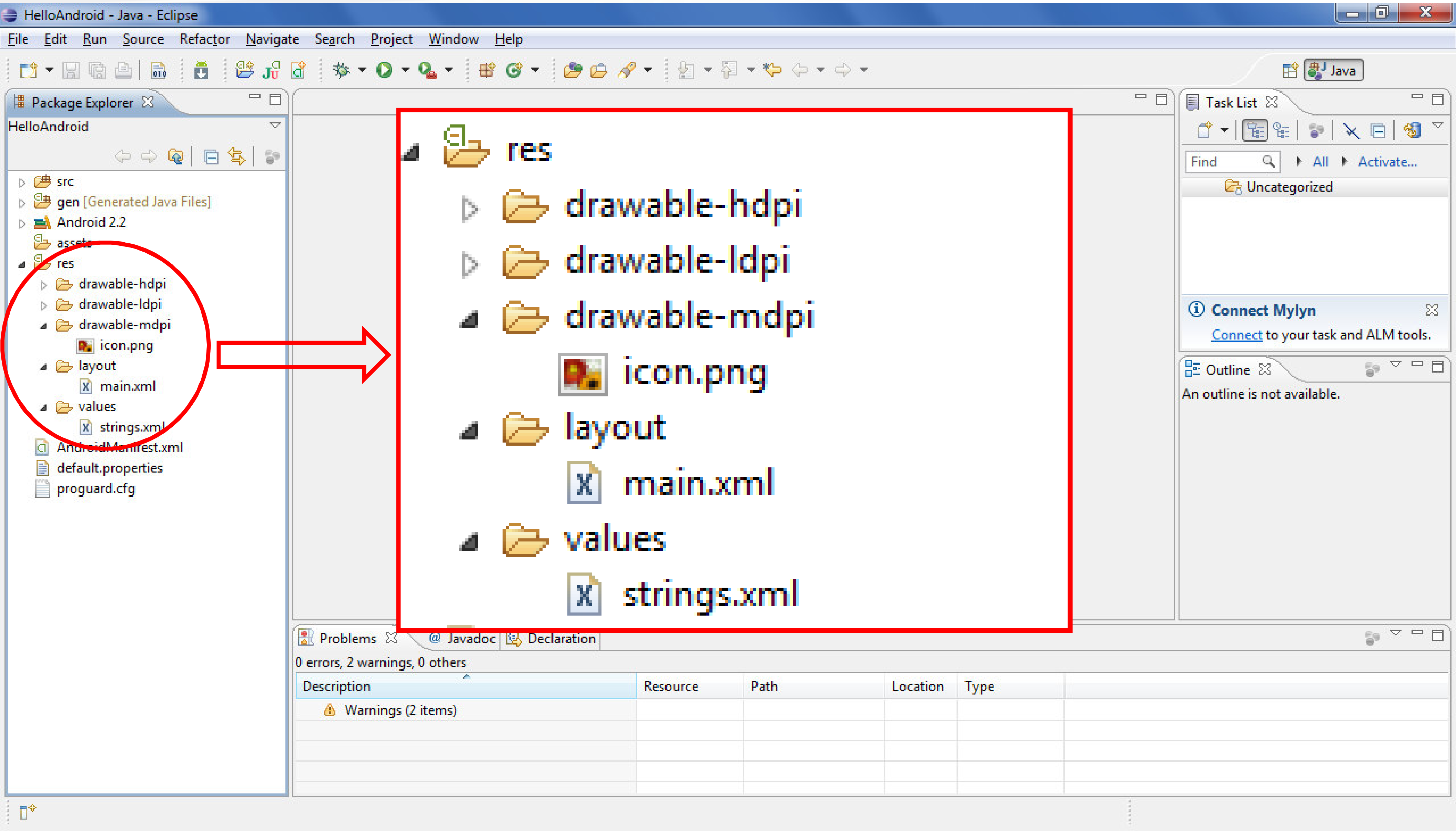
onPause is last state to preserve state and cleanup before app possibly destroyed

Apps move through states during lifecycle

Resources

- Definition: *A resource is a localized text string, bitmap, or other small pice of noncode information that a program needs*.
- All resouces get compiled into the application at build time.
- Resoures are created and stored in *res* directory inside a project.
- The resouce compiler compresses and packs all resouces and generates a class named R that contains identifiers to be referenced in a program.

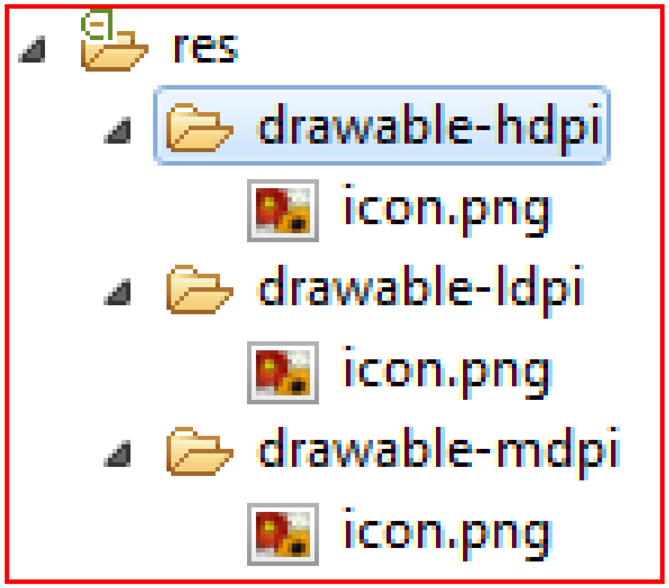
Resources

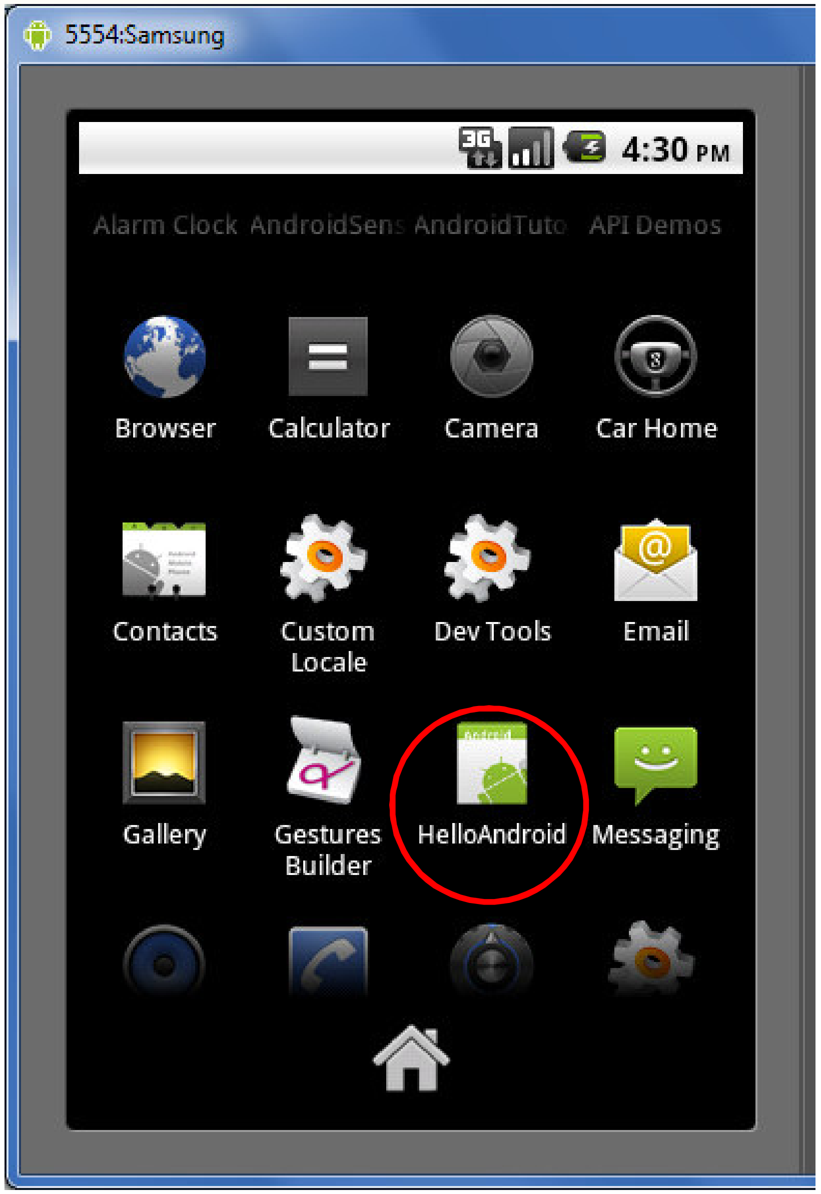


0 errors, 2 warnings, 0 others

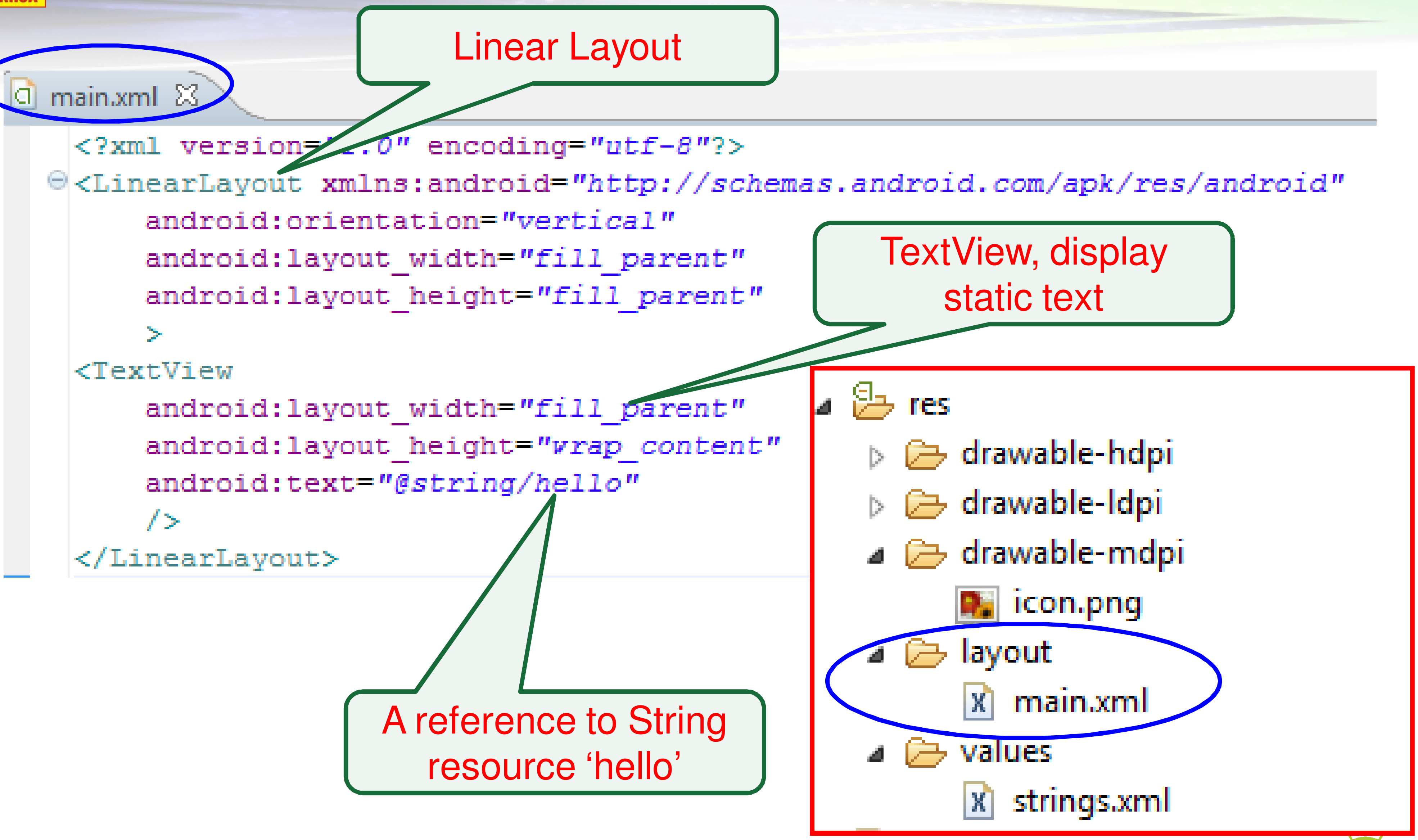
Description	Resource	Path	Location	Type
Warnings (2 items)				

Resources - icon


res
drawable-hdpi
icon.png
drawable-ldpi
icon.png
drawable-mdpi
icon.png

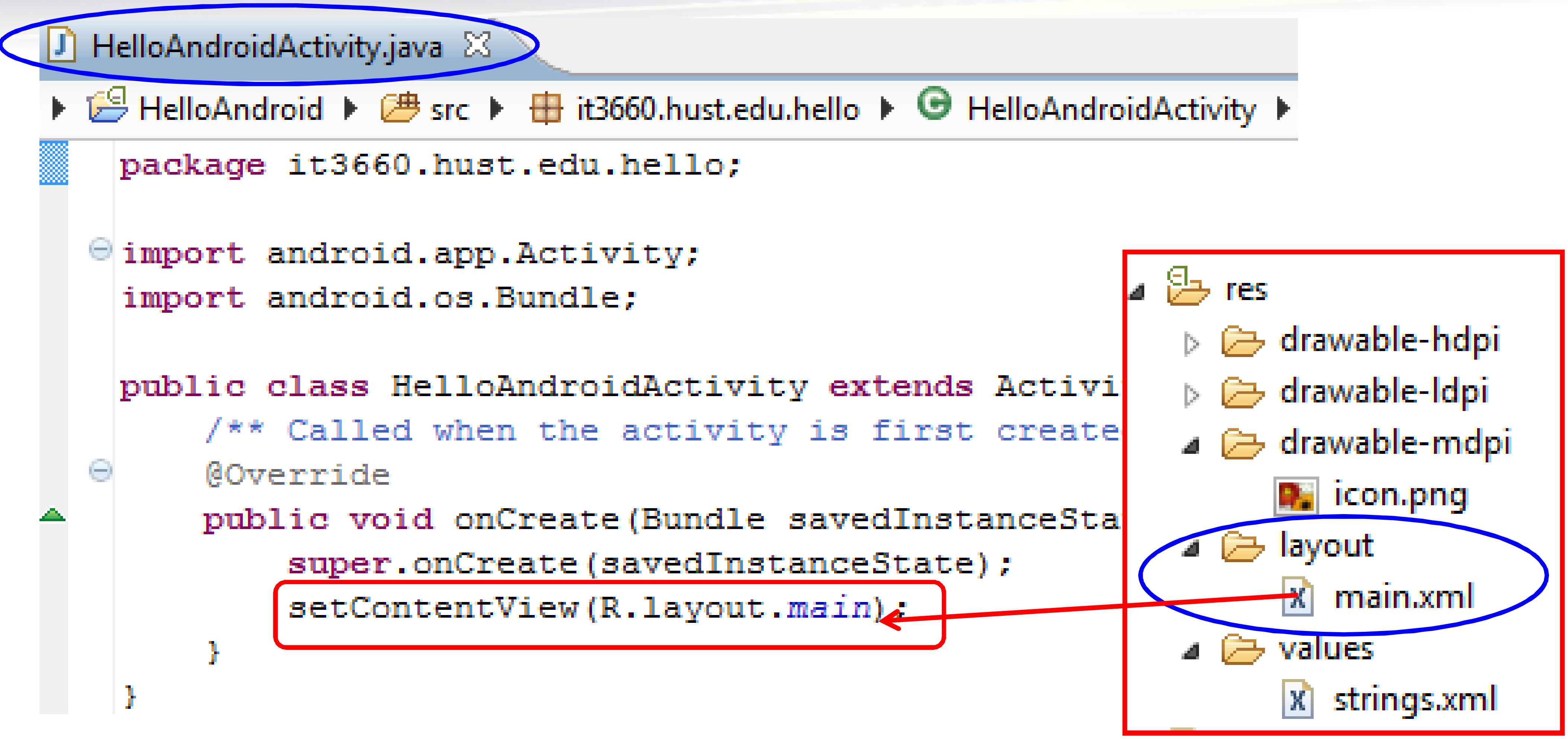


Resources - layout



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

Resources - layout

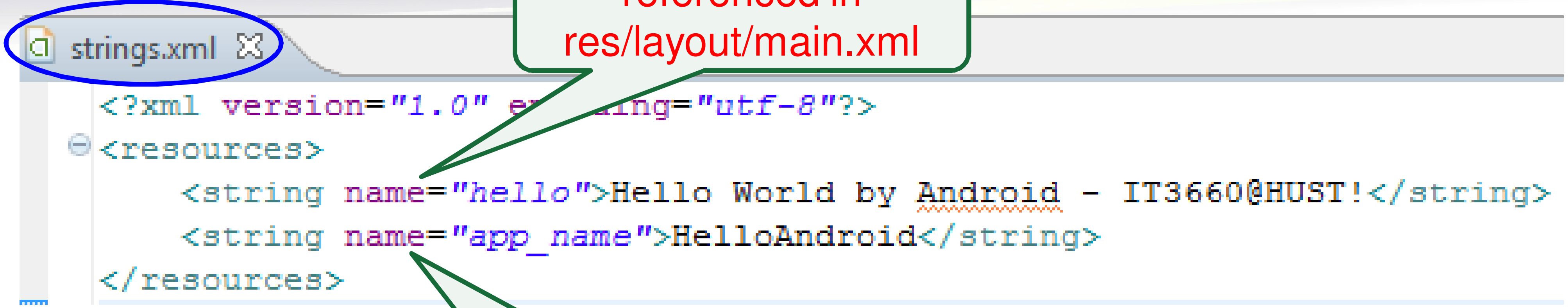


```
package it3660.hust.edu.hello;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroidActivity extends Activi
    /** Called when the activity is first create
    @Override
    public void onCreate(Bundle savedInstanceSta
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Resources - values

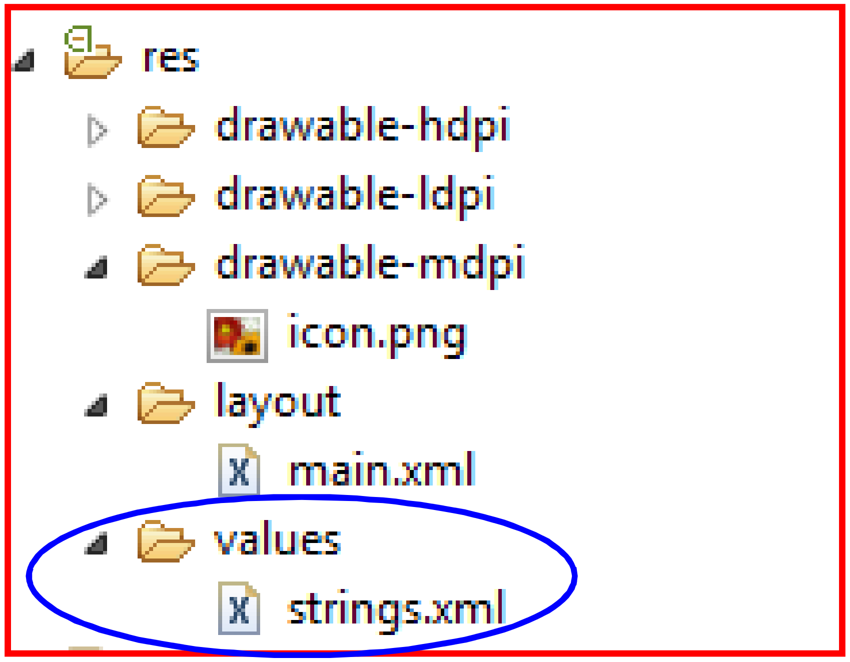


strings.xml

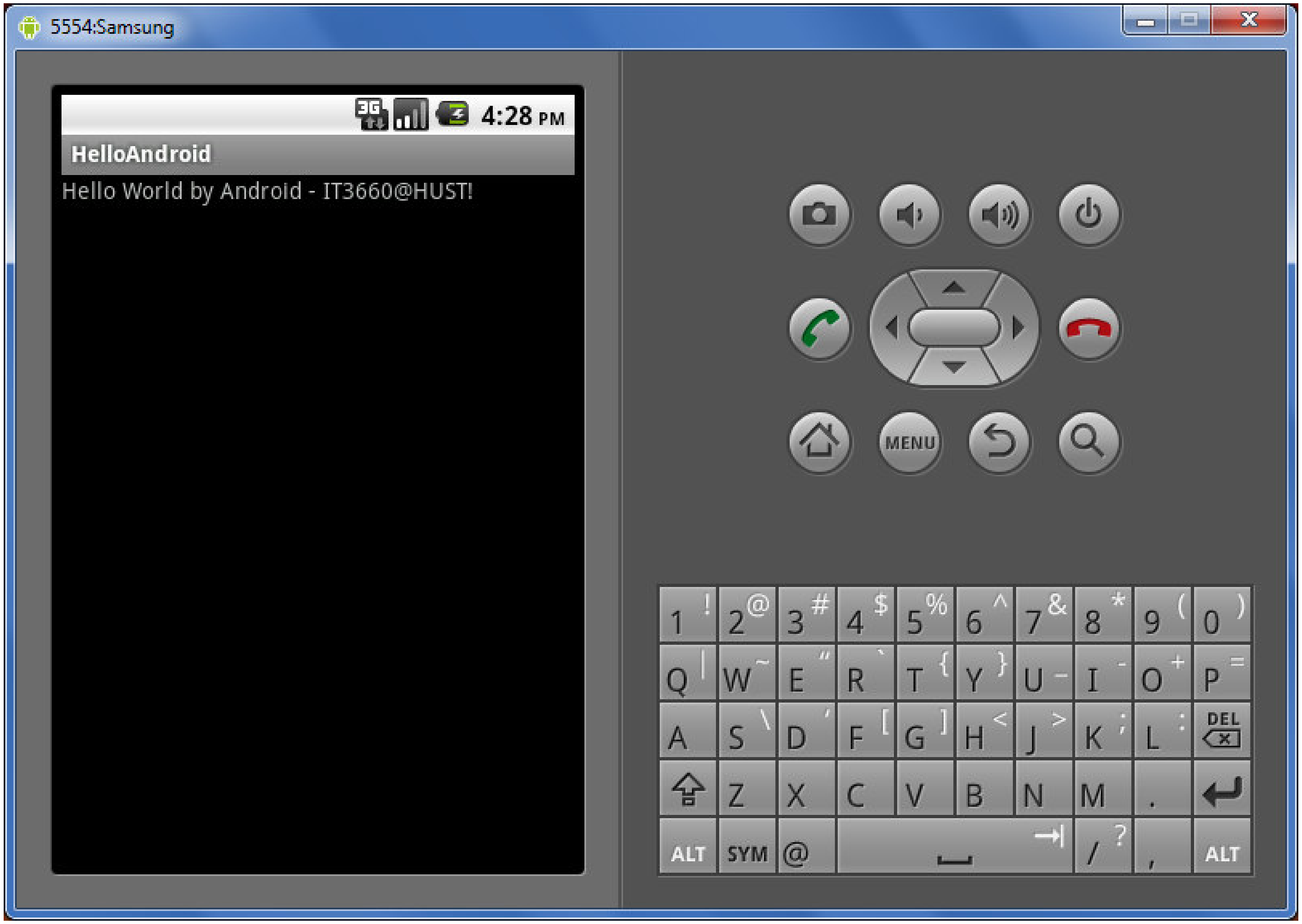
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World by Android - IT3660@HUST!</string>
    <string name="app_name">HelloAndroid</string>
</resources>
```

referenced in res/layout/main.xml

referenced in AndroidManifest.xml



Hello Android!



```
Hello World by Android - IT3660@HUST!
```

Resources

- You should always externalize resources such as images and strings from your application code, so that you can maintain them independently.
- Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations.
- In order to provide compatibility with different configurations, you must organize resources in your project's res/ directory, using various sub-directories that group resources by type and configuration.

Operations of Resources (1)

- **Providing Resources**

  What kinds of resources you can provide in your app, where to save them, and how to create alternative resources for specific device configurations.

- **Accessing Resources**

  How to use the resources you've provided, either by referencing them from your application code or from other XML resources.

Operations of Resources (2)

- **Handling Runtime Changes**

  How to manage configuration changes that occur while your Activity is running.

- **Localization**

  A bottom-up guide to localizing your application using alternative resources. While this is just one specific use of alternative resources, it is very important in order to reach more users.

- **Resource Types**

  A reference of various resource types you can provide, describing their XML elements, attributes, and syntax. For example, this reference shows you how to create a resource for application menus, drawables, animations, and more.

Resource types (1)

- **Animation Resources**
  - Define pre-determined animations.
  - Tween animations are saved in res/anim/ and accessed from the R.anim class.
  - Frame animations are saved in res/drawable/ and accessed from the R.drawable class.
- **Color State List Resource**
  - Define a color resources that changes based on the View state.
  - Saved in res/color/ and accessed from the R.color class.
- **Drawable Resources**
  - Define various graphics with bitmaps or XML.
  - Saved in res/drawable/ and accessed from the R.drawable class.

Resource types (2)

- **Layout Resource**
  - Define the layout for your application UI.
  - Saved in res/layout/ and accessed from the R.layout class.
- **Menu Resource**
  - Define the contents of your application menus.
  - Saved in res/menu/ and accessed from the R.menu class.
- **String Resources**
  - Define strings, string arrays, and plurals (and include string formatting and styling).
  - Saved in res/values/ and accessed from the R.string, R.array, and R.plurals classes.

Resource types (3)

- **Style Resource**
  - Define the look and format for UI elements.
  - Saved in res/values/ and accessed from the R.style class.
- **More Resource Types**
  - Define values such as booleans, integers, dimensions, colors, and other arrays.
  - Saved in res/values/ but each accessed from unique R sub-classes (such as R.bool, R.integer, R.dimen, etc.).

End of Lecture

Q&A