
# Mobile Application Development – Android

## Lecture 2

MTAT.03.262

Satish Srirama

satish.srirama@ut.ee

Mobile Cloud Lab

tiigriülikool

Programm IKT toetuseks kõrghariduses

Tiigriülikool on riiklik info- ja kommunikatsioonitehnoloogia- alane kõrghariduse toetusprogramm

# Android Lecture 1 - recap

- What is Android
- How to develop Android applications
- Run & debug the applications
- Using resources and discussed the advantages of structured resources
- Android application lifecycle

# Grading policy updates

- Up to 1 week delay
  - Your submission will be graded for 80%
- After 1 week delay until end of course
  - Your submission will be graded for 50%

# Outline

- Views and Layouts
- Events
- Basic application components

# Views

- Views are the building blocks
- The user interface is built using `View` and `ViewGroup` objects
- Examples:
  - Can be as basic as: `TextView, EditText, ListView`
  - Fancier views: `ImageView, MapView, WebView`

http://developer.android.com/resources/tutorials/views/index.html

# Simple View Items

- TextView
- EditText


  - Can also be used as a password field
- Button

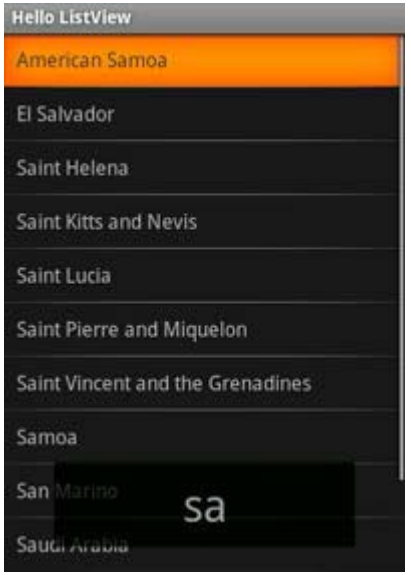

- CheckBox

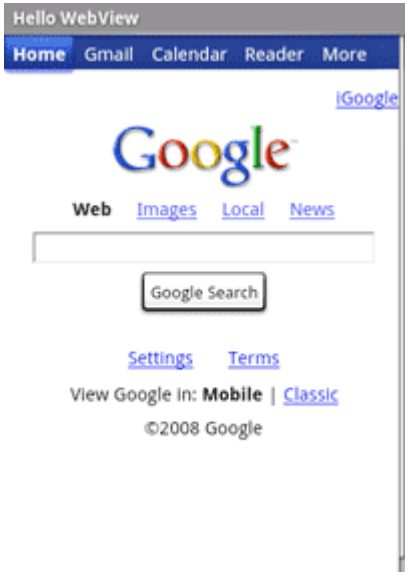
- RadioButton

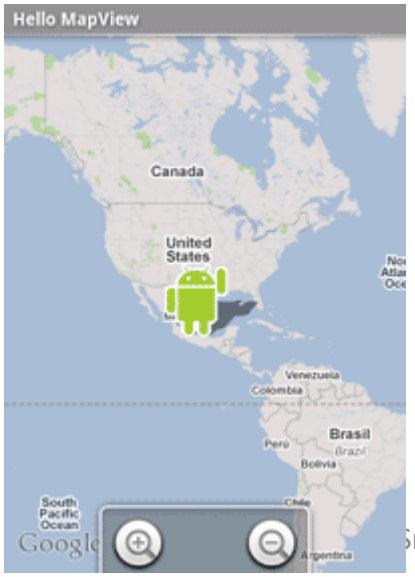
- Spinner

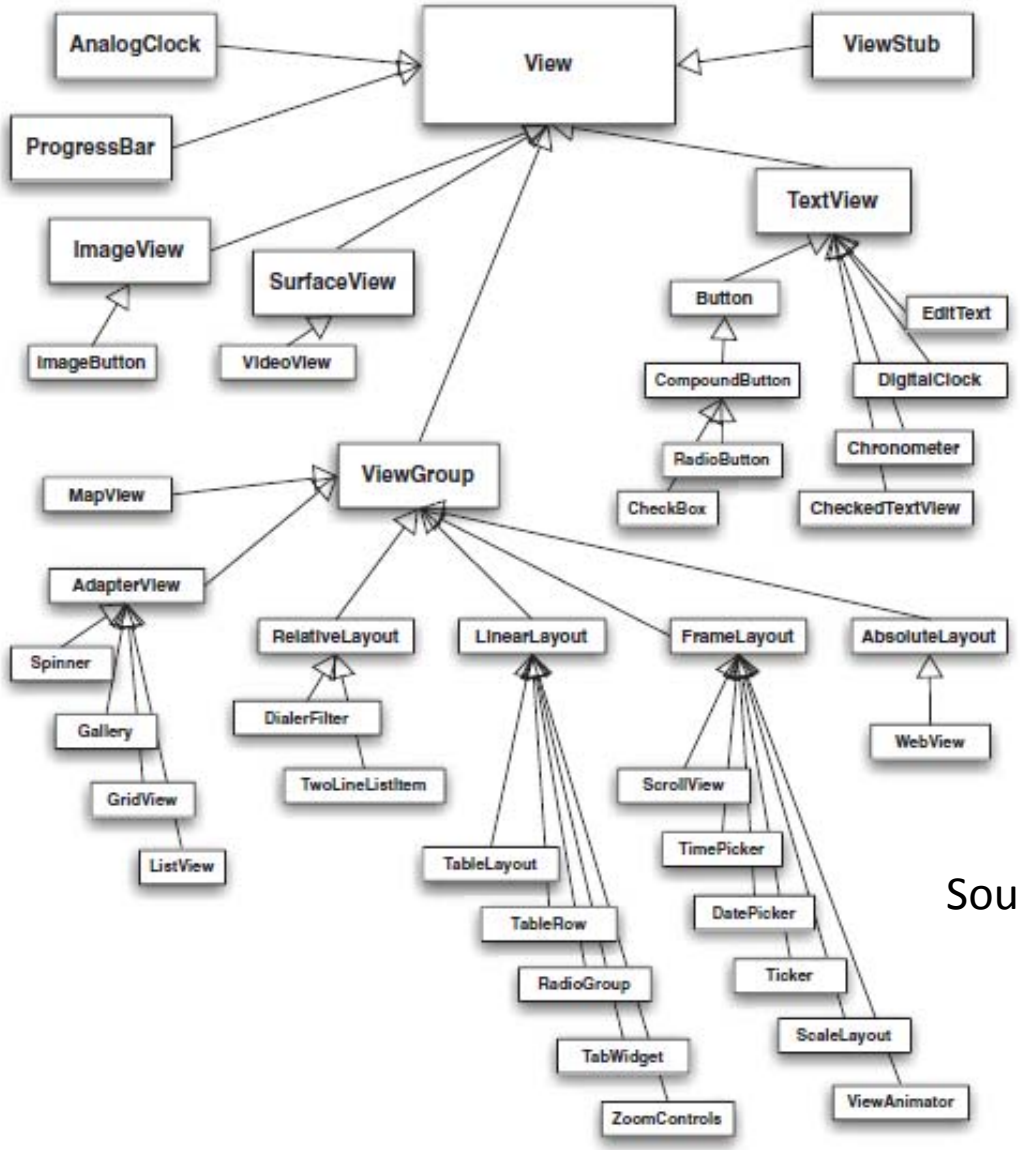
# More View Items

- ListView (ViewGroup)
- WebView
- MapView



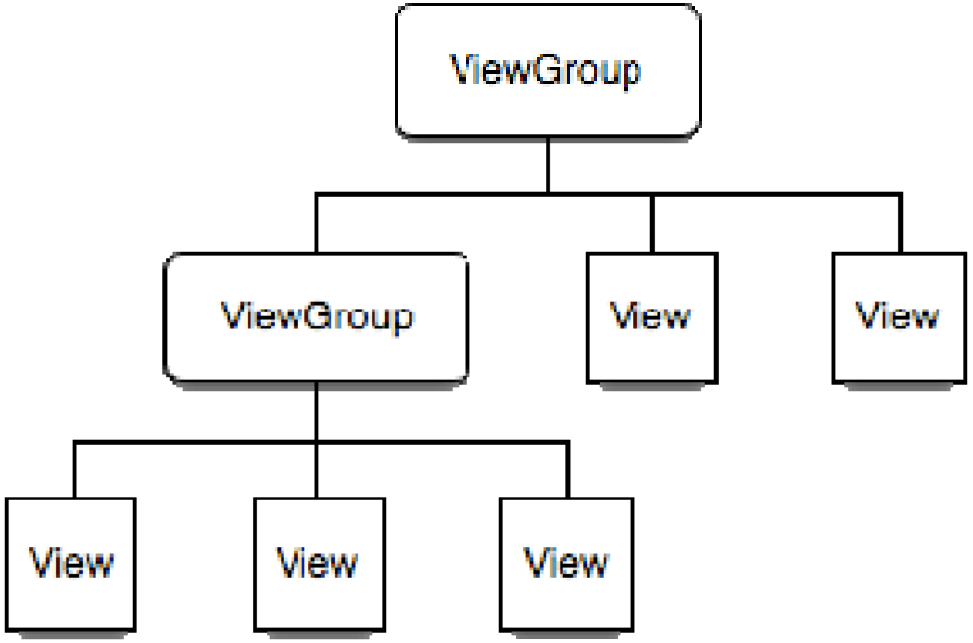



Source: unlocking android, p 71

**Figure 3.4 A class diagram of the Android `View` API, showing the root `View` class and specializations from there; notice that `ViewGroup` classes, such as layouts, are also a type of `View`.**

# View Hierarchy

- Activity's UI will be defined by a hierarchy of View and ViewGroup nodes
- `ViewGroup` objects are invisible view containers



# Layouts

- Layouts are ViewGroups which are used to hold other Views
- They are Invisible
- Allow positioning of different elements
- Layouts can be nested inside of each other

# Layouts

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

# Linear Layout

- Vertical: Makes one column of views
- Horizontal: Makes one row of views
- When a View (such as a Button) is added to a Layout, parameters can be set on how that View is considered within the Layout
  - FILL_PARENT: Expand the View as much as possible to fill the space of the parent container (Layout)
  - WRAP_CONTENT: Make the view be just large enough to hold its contents
  - Can apply to both width and height of View
- LAYOUT_WEIGHT: A weighting indicating relative sizes of multiple Views when sharing a layout

# Linear Layout - Example


```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">
    <TextView
        android:text="red"
        android:gravity="center_horizontal"
        android:background="#aa0000"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="green"
        android:gravity="center_horizontal"
        android:background="#00aa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="blue"
        android:gravity="center_horizontal"
        android:background="#0000aa"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="yellow"
        android:gravity="center_horizontal"
        android:background="#aaaa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
</LinearLayout>
```

# Exercise: Layouts

- Work with Linear Layout
  - Design Estonian and Italian national flags one after the other

http://developer.android.com/resources/tutorials/views/index.html

# RelativeLayout

- A Layout where the location for added Views can be described:
  - Relative to other Views added (“to the left of X”)
  - Relative to the RelativeLayout container (“aligned to the container bottom”)
- Very flexible
- Suggested for use over nested LinearLayouts
  - More complex the nesting of a layout, the longer to inflate

# RelativeLayout - Example


```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

# Input Events

- What are views without having support for user interactions!!!
- Define an event listener and register it with the View
  - `myButton.setOnClickListener(new View.OnClickListener() {});`
- Override an existing callback method for the View
  - `@Override`
    `public void onClick(View v)`

http://developer.android.com/guide/topics/ui/ui-events.html

# Input Events - continued

- Alternatively one can add the `android:onClick` attribute to the `<Button>` element.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

- Then define the `sendMessage` method in MainActivity

```
/** Called when the user clicks the Send button */
public void sendMessage(View view) {
    // Do something in response to button
}
```

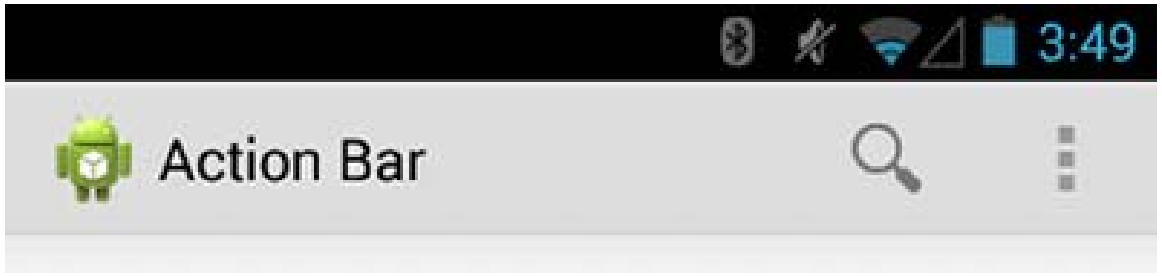
# Exercise: Input events

- Demo: How to use buttons

- Work with Relative Layout
- C to F converter

# Other interesting UI elements

- Action Bar
  - A dedicated space for giving your app an identity and indicating the user's location in the app
  - Access to important actions in a predictable way (such as Search)
  - Support for navigation and view switching (with tabs or drop-down lists)
- Beginning from Android 3.0 (API level 11), the action bar is included in all activities that use the Theme.Holo (default) theme
  - By default it contains - app icon and activity title




http://developer.android.com/training/basics/actionbar/index.html

# Options menu and action bar

- Primary collection of menu items for an activity
  - Place actions that have a global impact on the app
    - such as "Search," "Compose email," and "Settings."
  - On-screen action items and overflow options
- To define the menu, create an XML file inside your project's `res/menu/` directory

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

# Other interesting UI elements - continued

- Fragments
  - You can build dynamic UI
  - multi-pane user interface
- To create a fragment, extend the `Fragment` class
- Add the fragment to the activity layout using xml
  - <fragment android:name=„ee.ut.cs.mc.and.ArticleFragment"
    android:id="@+id/article_fragment„ android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="match_parent" />
- public class MainActivity extends FragmentActivity {


http://developer.android.com/training/basics/fragments/index.html

# Basic Application Components

- Application components are the essential building blocks of an Android application
- Activities
  - UI component
  - Typically corresponding to one screen.
- BroadcastReceivers
  - Respond to broadcast Intents
- Services
  - Faceless tasks that run in the background
- ContentProviders
  - Enable applications to share data

# Activities

- Typically correspond to one screen in a UI
- But, they can:
  - be faceless
  - be in a floating window
  - return a value
- For more info http://developer.android.com/guide/topics/fundamentals/activities.html

# Intents

- Unique concept in Android
- Mobile applications mostly run in their sandboxes
- Applications are isolated from each other
- Intents support interactions among the application components
  - Components can be of any application in the device
  - Android does not distinguish third party applications and native applications
- Activities, services, and broadcast receivers — are all activated through Intent messages

# Working with Intents

- Declare your intention that an Activity or Service be started to perform an action
- Broadcast that an event/action has occurred
- System matches Intent with Activity that can best provide that service
- Activities and BroadcastReceivers describe what Intents they can service in their IntentFilters (via AndroidManifest.xml)

# Using intents to launch activities

- Explicitly starting new Activities

```
Intent intent = new Intent(this,
MyOtherActivity.class);
startActivity(intent);
```

- The new activity is started and no connection with the current activity

- Demo GreetingActivity – (as part of course exercise)

# Intent Object

- An Intent object is a bundle of information
- Component name
  - The name of the component that should handle the intent
- Information of interest to the component that receives the intent
  - action
    - The general action to be performed
    - `ACTION_CALL` - Initiate a phone call
    - `ACTION_BATTERY_LOW` - A warning that the battery is low
  - Data
    - The URI of the data to be acted on
    - `Uri uri = Uri.parse("content://contact/people");`

http://developer.android.com/reference/android/content/Intent.html

# Examples of action/data pairs

- **ACTION_VIEW** ***content://contacts/people/1***
  - Display information about the person with identifier "1"
- **ACTION_DIAL** ***content://contacts/people/1***
  - Display the phone dialer with the person filled in
- **ACTION_VIEW** ***tel:123***
  - Display the phone dialer with the given number filled in
  - VIEW action does what is considered the most reasonable thing for a particular URI
- **ACTION_EDIT** ***content://contacts/people/1***
  - Edit information about the person whose identifier is "1"

http://developer.android.com/guide/components/intents-common.html

# Intent Object - continued

- category
  - Information of interest to the Android system
  - Category of component that should handle the intent
    - `CATEGORY_BROWSABLE` - The target activity can safely be invoked by the browser
    - `CATEGORY_LAUNCHER` - it should appear in the Launcher as a top-level application
  - ACTION_MAIN with category CATEGORY_HOME
    - Launch the home screen.

# Intent Object - continued

- extras
  - A Bundle of any additional information
  - can be used to provide extended information to the component
    - Example: if we have a action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("plain/text");
intent.putExtra(Intent.EXTRA_EMAIL,
   emailAddressList);
intent.putExtra(Intent.EXTRA_SUBJECT, emailSubject);
intent.putExtra(Intent.EXTRA_TEXT, emailText);
startActivity(intent);
```

# Implicit intents and late runtime binding

- The implicit Intents are mentioned in the Android Manifest file

```
<activity android:name=".ContactPickerActivity"
    android:label="@string/app_name">
    <!--
        <intent-filter> <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    -->
    <intent-filter>
        <action android:name="android.intent.action.PICK" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:path="contacts" android:scheme="content" />
    </intent-filter>
</activity>
```

# Intent resolution in Android

- Android puts together a list of Intent Filters
- Input Filters that do not match the action or category are removed from the list
- Each part of the Intent's data URI is compared to the Intent Filters data tag
- If more than one component is resolved, they are offered to the user to select

# Using intents to launch activities - continued

- To get the response back from the sub-Activity

```
private static final int SHOW_SUBACTIVITY = 1;
StartActivityForResult(intent,
SHOW_SUBACTIVITY);
```

- The response is sent back as

```
setResult(Activity.RESULT_OK, resultIntent);
```

# Homework

- Starting to use email
  - Create an intent to send mail from GreetingActivity
  - The list of ids you can put dynamically (I mean in the email screen itself – in *to* field)
  - Put all the necessary info
  - Then send the result back that the email is sent to the main activity
  - Display in main activity email is sent

# THANK YOU