Mobile Application Design and Development

CS 5520

Adrienne Slaughter

Fall 2021

Northeastern University
College of Computer and Information Science

App Starts
Functionality that needs permission
Check SDK Version
SDK < 23
SDK >= 23
Check if app has needed permission
User Already Granted Permission
Show Request Rationale
Display Why We Need Permission
Request Permission
Call Method
Handle Permission Request
Disable Functionality

ANDROID BASICS

Topics

- Debugging on Android
- Explain Android basics:
  - Activity lifecycle
  - Manifest file
  - Intents and intent filters
  - GUI: Layouts/views

Activity Lifecycle (…a bit more!)

- **Paused**

  Lost focus, but visible (e.g. transparent); retains state

- **Stopped**

  Completely obscured; retains state; often killed

Important: In either case, system can drop the activity by asking it to finish or by killing the process

Why so many states?

```
// Nice concept example from StackOverflow: DriveToWorkActivity (http://bit.ly/Wvv998)
protected void onCreate(...) {
    openGarageDoor();
    unlockCarAndGetIn();
    closeCarDoorAndPutOnSeatBelt();
    putKeyInIgnition();}

protected void onStart() {
    startEngine();
    changeRadioStation();
    switchOnLightsIfNeeded();
    switchOnWipersIfNeeded();}

protected void onResume() {
    applyFootbrake();
    releaseHandbrake();
    putCarInGear();
    drive();}
```

Why so many states?

```
protected void onPause() {
    putCarInNeutral();
    applyHandbrake();}

protected void onStop() {
    switchEveryThingOff();
    turnOffEngine();
    removeSeatBeltAndGetOutOfCar();
    lockCar();}

protected void onDestroy() {
    enterOfficeBuilding();}

protected void onReachedGroceryStore(...) {
    Intent i =
        new Intent(ACTION_GET_GROCERIES, ...,  this, GroceryStoreActivity.class);}

protected void onRestart() {
    unlockCarAndGetIn(); closeDoorAndPutOnSeatBelt(); putKeyInIgnition();}
```

Do only what you need when you need to do it because states change rapidly!

Activity lifecycle

- Lifetime
  - onCreate (setup “global” state)
  - onDestroy (release all major resources)
- Visible lifetime (“behind glass door”)
  - onStart (register BroadcastReceivers)
  - onStop (unregister)
- Foreground lifetime (frequent switching)
  - onResume (user interacting; lightweight code)
  - onPause (commit changes; lightweight code)

Typical activity

```
public class Activity extends ApplicationContext {
     protected void onCreate(Bundle savedInstanceState);

     protected void onStart();

     protected void onRestart();

     protected void onResume();

     protected void onPause();

     protected void onStop();

     protected void onDestroy();
 }
```

When does an activity change state?

- Configuration change
  - Portrait to Landscape
- Multiwindow mode (new!)
- Activity or dialog appears in foreground
- User taps Back button
- System kills app process

Honeycomb (3.0)

Method	Description	Killable?	Next
`onCreate()`	Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by `onStart()`.	No	`onStart()`
`onRestart()`	Called after your activity has been stopped, prior to it being started again. Always followed by `onStart()`	No	`onStart()`
`onStart()`	Called when the activity is becoming visible to the user. Followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden.	No	`onResume()` or `onStop()`
`onResume()`	Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by `onPause()`.	No	`onPause()`
`onPause()`	Called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either `onResume()` if the activity returns back to the front, or `onStop()` if it becomes invisible to the user.	**Pre-**`HONEYCOMB`	`onResume()` or `onStop()`
`onStop()`	Called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity is being started, an existing one is being brought in front of this one, or this one is being destroyed. Followed by either `onRestart()` if this activity is coming back to interact with the user, or `onDestroy()` if this activity is going away.	**Yes**	`onRestart()` or `onDestroy()`
`onDestroy()`	The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called `finish()` on it, or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the `isFinishing()` method.	**Yes**	*nothing*

Application Components

- Activities
  - Fragments
- Services
- Content Providers
- Broadcast receivers

Intents & Intent Filters

- Intents are used to:
  - Start an activity
  - Start a service
  - Deliver a broadcast

Why Intents?

- Supports abstraction
  - Your app can declare:
    - What kinds of things it can do
    - What kinds of things it wants to do
  - As a user, you can declare:
    - Which app you want to support a particular intent:
      - Google Calendar?
      - Acalendar?
      - Calengoo?
- Provides a safe, encapsulated means of communication between multiple apps

Manifest File

- Specifies the components that make up an app
- Identifies which Java class contains the code for a component
- Specifies which intents can be handled by a given component

Debugging: What did you do?

Debugging

- Goal:
  - Figure out what your app is doing or not doing, with the intent of making it do what you want.
- Approaches:
  - Ad-hoc
  - Print statements/ Logging
  - Debugger

Logging

- A structured approach to System.out.println();
- Structure includes:
  - Where the message originated
  - Level of severity of message
- Why this structure?
  - Ease of reading
  - Ease of filtering when reading
  - Ease of filtering when developing

Logging, Priority Levels

- **V** — Verbose (lowest priority)
- **D** — Debug
- **I** — Info
- **W** — Warning
- **E** — Error
- **A** — Assert

Adding Log statements in your code

- Using the **Log** class
  - `Log.e(String, String)` (error)
  - `Log.w(String, String)` (warning)
  - `Log.i(String, String)` (information)
  - `Log.d(String, String)` (debug)
  - `Log.v(String, String)` (verbose)
- The first param should provide info about where the message is originating from
- The second param should provide meaningful information

A good convention:

```
private static final String TAG = "MyActivity";
...
Log.i(TAG, "MyClass.getView() — get item number " + position);
```

The Logcat window

What is the User Interface?

- A set of components that:
  - present information to users,
  - allow users to input information,
- It's the thing that sits between the user and your logic

Big Picture View of Creating a UI

- Have an idea in mind of what it should look like and how it should act
- Decide on a layout/Layout Manager
- Decide on widgets
- Decide which widgets need to respond to user input
  - Which widgets the user will touch/interact directly with
- Write the code that does something when a widget does something
- Link the widget to the code
- Test, iterate.

Example: Assignment 1

- Sketch of app
- Layout: LinearLayout
- Widgets: Buttons
- OnClick(): Show a new activity
- Set the onClick listener on the buttons
- Run

UI Components

- Layouts
- Input Controls
- Input Events
- Dialogs
- Notifications
- Toasts
- App Bar
- …

Views & View groups

- All UI components extend a View or ViewGroup
  - Views: a UI component
  - ViewGroup: a UI component that holds other UI components

https://developer.android.com/guide/topics/ui/overview.html

Layouts

- Presenting information and *stuff* to the user
  - Grid
  - Frame
  - Linear
  - Relative
  - Constraint
  - Web

https://developer.android.com/guide/topics/ui/declaring-layout.html

Layout Parameters

- Specifies how each View relates to other Views
- The Layout Manager uses the LayoutParams to determine size & position of each View

https://developer.android.com/guide/topics/ui/declaring-layout.html

Northeastern University
College of Computer and Information Science

Building a Layout with an Adapter

Input Controls

- Getting input from the user
  - Button
  - Text field
  - Checkbox
  - Radio button
  - Toggle button
  - Spinner
  - Picker

https://developer.android.com/guide/topics/ui/controls.html#CommonControls

Input Events

- The process of getting input from users
- Input controls generate Input Events.
- Event Listeners:
  - onClick()
  - onLongClick()
  - onFocusChange()
  - onKey()
  - onTouch()
  - onCreateContextMenu()

Input Events, cont.

- Event Handlers:
  - onKeyDown()
  - onKeyUp()
  - onTrackballEvent()
  - onTouchEvent()
  - onFocusChanged()

- You write code that does something, overriding an event listener
- Create a UI component
- Assign an event listener to the UI component
- When that event happens on that UI component, your code is notified and run.

Dialogs

- Provide non-blocking information to the user
- Good for short feedback, errors, helpful info, etc.
- Example:

Notifications

- Provide information to the user outside of your app
- Examples:

Toasts

- Small message
- Only as big as it needs to be
- Fades in & out fairly quickly

Building the UI

- Decide on your Activities
- For each Activity, define your Layout & UI Components
- Build the UI in XML (or code)
- Write the Activity code that uses the layout defined in XML

Building the UI with XML

Inflating a layout (XML option)

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  setContentView(R.layout.main);
  TextView myTextView =
       (TextView)findViewById(R.id.myTextView);
}
```

Building the UI with Java

Inflating a layout (manual option)

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  TextView myTextView = new TextView(this);
  setContentView(myTextView);

  myTextView.setText("Hello, Android");
}
```

- Combining controls
- Adapting for different screen sizes and shapes

Resolution and density

- UIs running on a variety of screen sizes and densities
- Use resource qualifiers to deal with this “floodgate” of devices
- Potential hazard!

(They also have small changes in hardware and other quirks to keep in mind!)

Resolution and density

- Screen size
  - Small (smaller than 3.2” screen)
  - Medium (“Typical” size)
  - Large
- Pixel density
  - ldpi (100-140 dpi)
  - mdpi (140-180 dpi)
  - hdpi (190-250 dpi)
  - nodpi (no scaling regardless screen density)

Resolution and density

Aspect ratio

- long (significantly wider in landscape)
- notlong (typical ratio)

```
res/layout-small-long/ // Layouts for small, long
screens.
res/layout-large/      // Layouts for large screens.
res/drawable-hdpi/     // Drawabless for high density
screens.
```

To force screen size

- Use manifest

```
<supports-screens
  android:smallScreens="false"
  android:normalScreens="true"
  android:largeScreens="true"
  android:anyDensity="true"
/>
```

- anyDensity = false forces Android to scale

Tips

- Don't make assumptions about screens
- Avoid hard-coded pixel values
  - wrap_content
  - fill_parent
  - dp and sp
- Avoid AbsoluteLayout class
- RelativeLayout for complex UIs is probably what you need

Tips 2

Use Drawable resources rather than bitmaps

- NinePatches
- Shape Drawables
- Gradient Drawables
- Composite and transformable drawables
  - Rotate and Scale Drawables
  - LevelListDrawables
  - StateListDrawables

Asset that can't be scaled

- Image assess for each category to avoid aliasing/artifacts
  - res/drawable-ldpi
  - res/drawable-mdpi
  - res/drawable-hdpi
- Layouts for different phones
  - res/layout-small
  - res/layout-normal
  - res/layout-large

Test, test, test…

Using AVD you can define arbitrary screen resolutions and pixel densities

Modifying views

```
public class MyTextView extends TextView {

  public MyTextView (Context context, AttributeSet ats,
int defStyle) {
    super(context, ats, defStyle);
  }

  public MyTextView (Context context) {
    super(context);
  }

  public MyTextView (Context context, AttributeSet
attrs) {
    super(context, attrs);
  }

  @Override
  public void onDraw(Canvas canvas) {
    [ ... Draw things on the canvas under the text ...
]
```

Creating compound controls

```
public class MyCompoundView extends LinearLayout {
  public MyCompoundView(Context context) {
    super(context);
  }

  public MyCompoundV
attrs) {
    super(context, a
  }
}
```

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

Creating compound controls

```
public class ClearableEditText extends LinearLayout {

  EditText editText;
  Button clearButton;

  public ClearableEditText(Context context) {
    super(context);

    // Inflate the view from the layout resource.
    String infService =
Context.LAYOUT_INFLATER_SERVICE;
    LayoutInflater li;
    li =
(LayoutInflater)getContext().getSystemService(infServic
e);
    li.inflate(R.layout.clearable_edit_text, this,
true);

    // Get references to the child controls.
    editText = (EditText)findViewById(R.id.editText);
```

- Creating new controls

Completely new views

- Complete control over look and feel
- Extend View or SurfaceView classes
- View
  - Provides Canvas w/ draw methods
  - Paint classes
  - Use with bitmaps and rasters
- SurfaceView

  Surface object that supports openGL and drawing from background thread

Drawable resources

- XML definable (res/drawable folder)
- Types
  - ColorDrawable
  - ShapeDrawable
  - GradientDrawable (require gradient radius defined in pixels)
- Specify attributes with density independent pixels
- Run time scaled dyanamically

ColorDrawable

Solid color

```
<color
xmlns:android="http://schemas.android.com/apk/res/andro
id"
   android:color="#FF0000"
/>
```

ShapeDrawable

Shape by defining dimensions, background, stroke/outline

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://                Oval, rectangle, or ring          es/andro
id"
  android:shape="rectangle">
    <solid
      android:color="#f0600000"/>
    <stroke
      android:width="10dp"
      android:color="#00FF00"/>
    <corners
      android:radius="15dp" />
    <padding
      android:left="10dp"
      android:top="10dp"
```

GradientDrawable

Smooth transitions 2-3 colors in linear, radial, or sweep pattern

(sweep along outer edge of parent shape)

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/andro
id"
  android:shape="rectangle"
  android:useLevel="false">
  <gradient
    android:startColor="#ffffff"
    android:endColor="#ffffff"
    android:centerColor="#000000"
    android:useLevel="false"
    android:type="linear"
```

GradientDrawable

```
<!-- Oval with Radial Gradient -->
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/andro
id"
  android:shape="oval"
  android:useLevel="false">
  <gradient
    android:type="radial"
    android:startColor="#ffffff"
    android:endColor="#ffffff"
    android:centerColor="#000000"
    android:useLevel="false"
    android:gradientRadius="300"
  />
</shape>

<!-- Ring with Sweep Gradient -->
<?xml version="1.0" encoding="utf-8"?>
<shape
```

Gradient examples

http://escomic.net/217

Composite Drawables

- Combine bitmaps, shapes, and colors, and other composite drawables

- Four types
  - Transformative
  - Layer
  - State List
  - Level List

Transformative Drawables

- ScaleDrawable
- RotateDrawable

```
<!-- Rotation Drawable Resource -->
<?xml version="1.0" encoding="utf-8"?>
<rotate
xmlns:android="http://schemas.android.com/apk/res/andro
id"
  android:drawable="@drawable/icon"
  android:fromDegrees="0"
  android:toDegrees="90"
  android:pivotX="50%"
  android:pivotY="50%"
/>

<!-- Scale Drawable Resource -->
<?xml version="1.0" encoding="utf-8"?>
<rotate
```

Transformative Drawables

In code…

```
ImageView rotatingImage =
(ImageView)findViewById(R.id.RotatingImageView);
ImageView scalingImage =
(ImageView)findViewById(R.id.ScalingImageView);

// Rotate the image 50% of the way to it's final
orientation.
rotatingImage.setImageLevel(5000);

// Scale the image to 50% of it’s final size.
scalingImage.setImageLevel(5000);
```

Layer Drawables

Stack transparent images

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
xmlns:android="http://schemas.android.com/apk/res/andro
id">
  <item android:drawable="@drawable/bottomimage"/>
  <item android:drawable="@drawable/image2"/>
  <item android:drawable="@drawable/image3"/>
  <item android:drawable="@drawable/topimage"/>
</layer-list>
```

StateList Drawables

- Composite resource; specify a different drawable to display based on state of the View to which it is assigned
- **Most nati**
(e.g., buttons,

```
<selector
xmlns:android="http://schemas.android.com/apk/res/andro
id">
  <item android:state_window_focused="false"
        android:drawable="@drawable/widget_bg_normal"/>
  <item android:state_pressed="true"
       android:drawable="@drawable/widget_bg_pressed"/>
  <item android:state_focused="true"
      android:drawable="@drawable/widget_bg_selected"/>
  <item android:drawable="@drawable/widget_bg_normal"/>
</selector>
```

LevelList Drawables

- Overlay several specifying an integer index (useful for App Widgets)
- In code: imageView.setImageLevel(3)

```
<level-list
xmlns:android="http://schemas.android.com/apk/res/andro
id">
  <item android:maxLevel="0"
android:drawable="@drawable/im_0"/>
  <item android:maxLevel="1"
android:drawable="@drawable/im_1"/>
  <item android:maxLevel="2"
android:drawable="@drawable/im_2"/>
  <item android:maxLevel="4"
android:drawable="@drawable/im_4"/>
  <item android:maxLevel="6"
android:drawable="@drawable/im_6"/>
```

Keep in mind

- Inflating expensive
- Complex layouts confusing
  - For you
  - For user
- Avoid unnecessary nesting
- Avoid too many Views
- Avoid deep nesting
- To help: layoutopt command line tool (look it up)

Keep in mind, cont.

- Tempting to want to create custom views
  - Adds complexity
  - User is familiar with standards (perhaps apply a nice theme, but don't force them to learn something new)
- Best to extend appearance of existing views or combine views

Best Practices for UI

- Why?

Best Practices for UI

- Design for multiple screens
- Build a responsive UI with Constraint Layout
- Adding the AppBar
- Showing Popup Messages
- Creating custom views
- Creating backward-compatible UIs
- Implementing Accessibility
- Managing the System UI
- Creating Apps with Material Design

https://developer.android.com/training/best-ui.html

Google's Material Design

- https://material.io/guidelines/material-design/introduction.html
- Introduced with Android 21
- Goal:
  - Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.
  - Develop a single underlying system that allows for a unified experience across platforms and device sizes. Mobile precepts are fundamental, but touch, voice, mouse, and keyboard are all first-class input methods.
- Principles:
  - Material is the metaphor
  - Bold, graphic and intentional
  - Motion provide meaning

- Fragments vs ViewGroups

https://developer.android.com/guide/components/fragments.html

REVIEW

A person who never made a mistake never tried anything new. Albert Einstein

UI fundamentals

- Views
  - Controls or widgets (not App Widgets)
  - All UI controls, including layout classes, derived from views
- View Groups
  - Extension of View that can contain multiple child groups
  - ViewGroup extended to provide layout managers that help you layout controls
- Activities

Creating UI with views

Inflating a layout (XML option)

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  setContentView(R.layout.main);
  TextView myTextView =
(TextView)findViewById(R.id.myTextView);
}
```

Creating UI with views

Inflating a layout (manual option)

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  TextView myTextView = new TextView(this);
  setContentView(myTextView);

  myTextView.setText("Hello, Android");
}
```

Widget types

- TextView

  Multiline, string formatting, auto word wrap

- EditText

  Multiline entry, word wrap, hints

- ListView

  Vertical list of views

- Spinner

  Text view and associated List View; select an item from a list to display in the textbox

Widget types

- Button
- CheckBox
- RadioButton
- ViewFlipper

  Collection of views as a horizontal row where only one view is visible at a time; animated transitions
- QuickContactBadge

  Badge w/ image icon assigned to contact

Info on views

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

Layout managers (aka layouts)

- Extensions of ViewGroup
- Can be nested for arbitrary complexity
- But, watch out! Behavior can be a little tricky with fancy nesting

- Useful website:

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

Common layouts

- FrameLayout

  Pins children to top left corner

- LinearLayout

  Vertical or horizontal line

- RelativeLayout

  Flexible. Define positions relative to others and screen boundaries

- TableLayout

  Grid of rows and columns

XML for layouts

- Decouple presentation from View/Activity
- Hardware-specific variations dynamically loaded

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
  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="Enter Text Below"
  />
  <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Text Goes Here!"
  />
</LinearLayout>
```

In code an option

- Avoid where you can
- But sometimes you'll need

```
LinearLayout ll = new LinearLayout(this);
ll.setOrientation(LinearLayout.VERTICAL);

TextView myTextView = new TextView(this);
EditText myEditText = new EditText(this);

myTextView.setText("Enter Text Below");
myEditText.setText("Text Goes Here!");

int lHeight =
LinearLayout.LayoutParams.FILL_PARENT;
int lWidth =
LinearLayout.LayoutParams.WRAP_CONTENT;

ll.addView(myTextView, new
LinearLayout.LayoutParams(lHeight, lWidth));
ll.addView(myEditText, new
LinearLayout.LayoutParams(lHeight, lWidth));
setContentView(ll);
```