Lec #2: Activities, Intents, and User Interface
Content

• Review
• Android Activities
• Android Intents
• User Interface / Views
• User Notifications
Review
Developing for Android

- Eclipse
- Android SDK
- Android Development Tools (ADT)
- Android Virtual Devices (AVD) & SDK Manager
- The Android Emulator
- Dalvik Debug Monitor Services (DDMS)
- The Android Debug Bridge (ADB)
Android Software Stack

- **Applications**
- Home
- Contacts
- Phone
- Browser
- ...

- **Application Framework**
  - Activity Manager
  - Window Manager
  - Content Providers
  - View System
  - Package Manager
  - Telephony Manager
  - Resource Manager
  - Location Manager
  - Notification Manager

- **Libraries**
  - Surface Manager
  - Media Framework
  - SQLite
  - OpenGL ES
  - FreeType
  - WebKit
  - SGL
  - SSL
  - libc

- **Android Runtime**
  - Core Libraries
  - Dalvik Virtual Machine

- **Linux Kernel**
  - Display Driver
  - Camera Driver
  - Flash Memory Driver
  - Binder (IPC) Driver
  - Keypad Driver
  - WiFi Driver
  - Audio Drivers
  - Power Management
Your First Android Project

• Select File -> New -> Project… -> Android -> Android Project and create the Android project

• right-click the project and select Run As > Android Application
Project Structure

- **src/** - Java packages. Each package can have multiple .java files representing different classes.

- **res/layout/** - XML files that specify the layout of each screen.

- **res/values/** - XML files used as references by other files.

- **res/drawable-hdpi/**, **res/drawable-mdpi/**, and **res/drawable-ldpi/** - high, medium, and low dots-per-inch resolution pictures.

- **res/color**, **res/menu**, **res/anim**

- **assets/** - additional non-media files.

- **AndroidManifest.xml** specifies the project to the Android OS.

- **Auto-generated files (do not modify):**
  - **gen/** contains auto-generated code. Class **R.java** generated by **Android Asset Packaging Tool** (**aapt**).
  - **default.properties** contains project settings.
Project Structure
Android Component Model

Packaging: APK File (Android Package)
> Collection of components
> Components share a set of resources
  – Preferences, Database, File space
> Components share a Linux process
  – By default, one process per APK
> APKs are isolated
  – Communication via Intents or AIDL
> Every component has a managed lifecycle

Task (what users know as applications)
> Collection of related activities
> Capable of spanning multiple processes
> Associated with its own UI history stack
> Processes are started & stopped as needed
> Processes may be killed to reclaim resources
Android Application Components

i) Activity: Activity is a visual screen for interaction of user with the application. Depends upon design, an application may consists of one or more activities.

ii) Views: The User interface of an Activities is build with widgets.

ii) Service: Service do not have a visual interface, it runs in the back ground, like play back music and fetching data from the network.

iii) Broadcast Receiver: Broadcast receiver receive broadcast announcements and response to them according to the situation.

iv) Content Provider: Content provider is a SQLite database, which supports the sharing and accessing of data among applications.

v) Intents: Asynchronous messages which allow the application to request functionality from other services or activities.

vi) Others parts are Android widgets/Live Folders and Live Wallpapers.
Activities*

Android’s Activity

• An "activity" is an application component that provides a screen with which users can interact

• **Activity is usually a single screen**
  - Implemented as a single class extending Activity
  - Displays user interface controls (views)
  - Reacts on user input / events

• **An application typically consists of several activities**
  - Each screen is typically implemented by one activity
  - Each activity can then start another activity (new screen)
  - An activity may return a result to the previous activity

• "**main**" activity is presented to the user when launching the application for the first time.

• Each activity receives callbacks due to a change in its state during its **lifecycle** — whether the system is creating it, stopping it, resuming it, or destroying it.
Activity Lifecycle

Starting

- (1) onCreate()
- (2) onStart()
- (3) onRestoreInstanceState() *
- (4) onResume()

Running

- (1) onSaveInstanceState() *
- (2) onPause()
- (3) onResume()

Paused

- (1) onSaveInstanceState() *
- (2) onStop()

Stopped

- onDestroy() or <Process killed>

Destroyed

* (optional)
Activity Lifecycle

**Foreground Lifetime**
- Activity launched
- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()

**Visible Lifetime**
- App process killed

**Entire Lifetime**
- User navigates to the activity
- User returns to the activity
- User navigates to the activity
- Apps with higher priority need memory
- The activity is no longer visible
- The activity is finishing or being destroyed by the system

- onRestart()
public class Activity extends ApplicationContext {

    protected void onCreate(Bundle savedInstanceState);
    // Called when the activity is first created. Provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by onStart().

    protected void 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.

    protected void onRestart();
    // Called after your activity has been stopped, prior to it being started again. Always followed by onStart()

    protected void 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().

    protected void onPause();
    // Called when the system is about to start resuming a previous activity. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either onResume() or onStop().

    protected void onStop();
    // Called when the activity is no longer visible to the user. Followed by either onRestart() if this activity is coming back to interact with user, or onDestroy() if this activity is going away.

    protected void 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.
}
Utilizing Activity Lifecycle Functions

- Simple exercise to see the activity lifecycle in action
- each overridden function is explicit and a Toast command is added to show on screen when the function is entered
- Run it on an Android device and try various cases:
  - Changing the screen orientation destroys and recreates the activity from scratch.
  - Pressing the Home button pauses the activity, but does not destroy it.
  - Pressing the Application icon might start a new instance of the activity, even if the old one was not destroyed.
  - Letting the screen sleep pauses the activity and the screen awakening resumes it. (This is similar to taking an incoming phone call.)

```java
package org.example.hello;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class Hello extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onResume() {
        super.onResume();
        Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onPause() {
        Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
        super.onPause();
    }

    @Override
    protected void onStop() {
        Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
        super.onDestroy();
    }
}
```
Activity Life Cycle Samples

• Turn Display
  > onCreate(null) -> onStart -> onResume() -> [Turn Display]
  > onSaveInstanceState() -> onPause() -> onStop() -> onDestroy()
    -> onCreate() -> onStart() -> onRestoreInstanceState() -> onResume()

• Home
  > onCreate(null) -> onStart -> onResume() -> [Home Button]
  > onSaveInstanceState() -> onPause() -> onStop() -> [Start App]
  > onRestart() -> onStart() -> onResume()

• Phone Call Interrupt
  > onCreate(null) -> onStart -> onResume() -> [Phone Call]
  > onSaveInstanceState() -> onPause() -> onStop() -> [Hang Up or press Back]
  > onRestart() -> onStart() -> onResume()
Notes on Activity

• Starting Activity:
  • void **startActivity** (Intent intent)
  • void **startActivityForResult** (Intent intent, int requestCode)
    • void **onActivityResult** (int requestCode, int resultCode, Intent data)

• Shutting Down Activity:
  • void **finish** ()
  • void **finishActivity** (int requestCode)

• Saving Activity Status
  • void **onSaveInstanceState** (Bundle outState)
  • void **onRestoreInstanceState** (Bundle savedInstanceState)

• Activity A starts Activity B:
  • A's **onPause()**
  • B's **onCreate()**, **onStart()**, and **onResume()** (Activity B now has user focus.)
  • A’s **onStop()**, if A is no longer visible
public class MyActivity extends Activity {

    ... 

    static final int PICK_CONTACT_REQUEST = 0;

    protected boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
            // When the user center presses, let them pick a contact.
            startActivityForResult(
                new Intent(Intent.ACTION_PICK,
                            new Uri("content://contacts"),
                            PICK_CONTACT_REQUEST);
            return true;
        }
        return false;
    }

    protected void onActivityResult(int requestCode, int resultCode,
                                      Intent data) { 
        if (requestCode == PICK_CONTACT_REQUEST) {
            if (resultCode == RESULT_OK) {
                // A contact was picked. Here we will just display it
                // to the user.
                startActivity(new Intent(Intent.ACTION_VIEW, data));
            }
        }
    }
}
public class CalendarActivity extends Activity {

    ...

    static final int DAY_VIEW_MODE = 0;
    static final int WEEK_VIEW_MODE = 1;

    private SharedPreferences mPrefs;
    private int mCurViewMode;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        SharedPreferences mPrefs = getSharedPreferences();
        mCurViewMode = mPrefs.getInt("view_mode", DAY_VIEW_MODE);
    }

    protected void onPause() {
        super.onPause();

        SharedPreferences.Editor ed = mPrefs.edit();
        ed.putInt("view_mode", mCurViewMode);
        ed.commit();
    }
}
Back Stack

Tasks and Activities

Multiple Tasks

Multiple Activity Instances
Activities and AndroidManifest.xml

- Any new activity must be defined in AndroidManifest.xml file
Intents*

* [http://developer.android.com/reference/android/content/Intent.html](http://developer.android.com/reference/android/content/Intent.html)
Intents

• Intent

- Intents are used as a message-passing mechanism within your application and between applications.
  - Explicit start Activity/Service using its class name
  - Start Activity/Service for an action with specific data
  - Broadcast that an event has occurred
- Consists of
  - Component name (name of component/activity to handle the intent)
  - Action to be performed (MAIN / VIEW / EDIT / PICK / DELETE / …)
  - Data to act on (expressed as URI)
  - Categories
  - Extras (primitives / primitives / Strings / Serializable)
  - Flags

```java
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.fhnw.ch"));

Intent intent = new Intent(Intent.ACTION_EDIT);
intent.setData(Uri.parse("content://contacts/people/1"));
```
Component Name Field

• Specifies the name of the component (name of the activity if the component is activity) that should handle the intent

  • Class name of the target component (for example "FirstActivity")

    `<activity android:name=".FirstActivity"
              android:label="@string/app_name">
        <intent-filter>
          <action android:name="edu.odu.examples.IntentActivity" />
          <category
              android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>`

• Setting component name is optional

  • If it is set, the Intent object is delivered to an instance of the designated class.

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

    **Explicit Intent**

  • If it is not set, Android uses other information (i.e., action field) in the Intent object to locate a suitable target - this is called “intent resolution”

    `Intent intent = new Intent("edu.odu.examples.IntentActivity");`

    **Implicit Intent**
Action Field

• A string naming the action to be performed

• The Intent class defines a number of predefined action constants, including
  
  • ACTION_CALL, ACTION_EDIT, ACTION_MAIN, ACTION_SYNC, ACTION_BATTERY_LOW, etc.

• You can also define your own action strings for activating the components in your application

• The action largely determines how the rest of the intent is structured - particularly the data and extras fields - much as a method name determines a set of arguments and a return value.
## Intent Actions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Target component</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_CALL</td>
<td>activity</td>
<td>Initiate a phone call.</td>
</tr>
<tr>
<td>ACTION_EDIT</td>
<td>activity</td>
<td>Display data for the user to edit.</td>
</tr>
<tr>
<td>ACTION_MAIN</td>
<td>activity</td>
<td>Start up as the initial activity of a task, with no data input and no returned output.</td>
</tr>
<tr>
<td>ACTION_SYNC</td>
<td>activity</td>
<td>Synchronize data on a server with data on the mobile device.</td>
</tr>
<tr>
<td>ACTION_BATTERY_LOW</td>
<td>broadcast receiver</td>
<td>A warning that the battery is low.</td>
</tr>
<tr>
<td>ACTION_HEADSET_PLUG</td>
<td>broadcast receiver</td>
<td>A headset has been plugged into the device, or unplugged from it.</td>
</tr>
<tr>
<td>ACTION_SCREEN_ON</td>
<td>broadcast receiver</td>
<td>The screen has been turned on.</td>
</tr>
<tr>
<td>ACTION_TIMEZONE_CHANGED</td>
<td>broadcast receiver</td>
<td>The setting for the time zone has changed.</td>
</tr>
</tbody>
</table>

Note that Actions’ “Constant” could be replaced with an equivalent String constant.

 e.g.,: ACTION_CALL = "android.intent.action.CALL"
Data Field

• The URI of the data to be acted on and the MIME type of that data.

• Different actions are paired with different kinds of data specifications.
  
  • If the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing.

  • If the action is ACTION_CALL, the data field would be a tel: URI with the number to call.

  • If the action is ACTION_VIEW and the data field is an http: URI, the receiving activity would be called upon to download and display whatever data the URI refers to.

• Examples of Action/Data Pairs
  
  • ACTION_VIEW content://contacts/people/ -- Display a list of people to browse through.

  • ACTION_VIEW content://contacts/people/1 – Display information about person whose id is "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.

  • ACTION.Dial tel:123 -- Dial the phone dialer with the given number filled in.
Launching Activity

• Intent Invocation

  ➢ Launch new activity (without receiving a result)

      startActivity(intent);

  ➢ Launch new activity and expect result

      startActivityForResult(intent, requestCode)

          – When activity exits, onActivityResult() method is called with given
            requestCode (int > 0)

      onActivityResult(requestCode, resultCode, result)
Intents and Intent Filters

• Intent Filters
  > Description of what intents an activity can handle
  > Activities publish their intent filters in a manifest file

<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <category android:name="android.intent.category.BROWSABLE"/>
  <data android:scheme="geo"/>
</intent-filter>

➢ Upon invocation of startActivity(intent) the system looks at the intent filters of all installed applications
  ➢ Candidate activity that owns the filter must pass matching tests for action, category, and data fields

Example of Actions

• Starting Activity:
  • void **startActivity** (**Intent** intent)

    Intent i = new **Intent** (this, OtherActivity.**class**);
    startActivity(i);

  • void **startActivityForResult** (**Intent** intent, int requestCode)
    • void **onActivityResult** (int requestCode, int resultCode, **Intent** data)

• Shutting Down Activity:
  • void **finish** ()

    Intent intent = new **Intent**(this, FirstActivity.**class**);
    startActivity(intent);
Activities that can initiate applications have filters with "android.intent.action.MAIN" specified as the action
  • This is a way an application gets started fresh, without a reference to any particular data.

If they are to be represented in the application launcher, they also specify the "android.intent.category.LAUNCHER" category:

```xml
<intent-filter . . . >
  <action android:name="code android.intent.action.MAIN" />
  <category android:name="code android.intent.category.LAUNCHER" />
</intent-filter>
```

The Android system populates the application launcher by finding all the activities with intent filters that specify the "android.intent.action.MAIN" action and "android.intent.category.LAUNCHER" category. It then displays the icons and labels of those activities in the launcher.
User Interface & Views

Contents

• Views and View Groups
• Views (Widgets)
• Layouts
• Events
• Menus
• Dialogs
• Styles and themes
Fundamental UI Design

• **Views**
  – Visual interface element (controls or widgets)

• **ViewGroup**
  – Contains multiple widgets. Layouts inherit the ViewGroup class

• **Activities**
  – Screen being displayed to the user. Assign a View or layout to an Activity
Layouts*

• Layout contains Views

• FrameLayout
  – Each child view is in reference to the top left corner. Use if only one View in screen at a time.

• LinearLayout
  – Views in line, either vertically or horizontally

• RelativeLayout
  – Define positions of each other child view relative to each other and screen boundaries

• TableLayout
  – Rows and Columns

• Web View
  – Display web pages

Layouts Examples

Linear Layout

Relative Layout

Grid Layout

Tab Layout
Defining Layout

• Portrait mode:
  – res/layout/main.xml

• Landscape mode:
  – res/layout-land/main.xml

• Each Activity can have its own layout

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

```xml
<?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="I am a TextView"/>
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button"/>
</LinearLayout>
```
## Common Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>layout_width</td>
<td>Specifies the width of the View or ViewGroup</td>
<td></td>
</tr>
<tr>
<td>layout_height</td>
<td>Specifies the height of the View or ViewGroup</td>
<td></td>
</tr>
<tr>
<td>layout_marginTop</td>
<td>Specifies extra space on the top side of the View or ViewGroup</td>
<td></td>
</tr>
<tr>
<td>layout_marginBottom</td>
<td>Specifies extra space on the bottom side of the View or ViewGroup</td>
<td></td>
</tr>
<tr>
<td>layout_marginLeft</td>
<td>Specifies extra space on the left side of the View or ViewGroup</td>
<td></td>
</tr>
<tr>
<td>layout_marginRight</td>
<td>Specifies extra space on the right side of the View or ViewGroup</td>
<td></td>
</tr>
<tr>
<td>layout_gravity</td>
<td>Specifies how child Views are positioned</td>
<td>Only in LinearLayout or TableLayout</td>
</tr>
<tr>
<td>layout_weight</td>
<td>Specifies the ratio of Views</td>
<td>Only in LinearLayout or TableLayout</td>
</tr>
</tbody>
</table>
Example of Linear Layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:stretchcolumns="*"
>
    <Button
        android:layout_width="100px"
        android:layout_height="wrap_content"
        android:layout_weight="0.1"
        android:text="Button A"
    />
    <Button
        android:layout_width="100px"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:layout_weight="0.3"
        android:text="Button B"
    />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_weight="0.6"
    />
</LinearLayout>
```
Example of Table Layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:background="#646464">
    <TableRow>
        <TextView android:text="E-mail:" android:width="120px" />
        <EditText android:id="@+id/email" android:width="200px" />
    </TableRow>
    <TableRow>
        <TextView android:text="Password:" /> 
        <EditText android:id="@+id/password" android:password="true" />
    </TableRow>
    <TableRow>
        <CheckBox 
            android:id="@+id/rememberMe" android:text="Remember Me" />
    </TableRow>
    <TableRow>
        <Button android:id="@+id/signIn" android:text="Log In" android:layout_span="2" />
    </TableRow>
</TableLayout>
```
Defining Styles

• Multiple Buttons

• Define style only once!

...<TableRow android:id="@+id/TableRow01"
    android:layout_weight="0.2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <Button android:id="@+id/Button07"
        style="@style/mybutton" android:text="7" />
    <Button android:id="@+id/Button08"
        style="@style/mybutton" android:text="8" />
    <Button android:id="@+id/Button09"
        style="@style/mybutton" android:text="9" />
    <Button android:id="@+id/ButtonDivide"
        style="@style/mybutton" android:text="/" />
</TableRow>
...
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="mybutton" parent="@android:style/TextAppearance">
    <item name="android:textSize">30sp</item>
    <item name="android:textColor">#000000</item>
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">fill_parent</item>
    <item name="android:layout_weight">1</item>
</style>
</resources>
Styles and Themes

• Way of building and formatting layout to your app
• Style has formatting attributes to several elements
• Theme has formatting attributes to all activities
  – Create the theme in xml (styles.xml)
  – Apply the theme in manifest

```xml
<application android:theme="@style/CustomTheme">
  <activity android:theme="@android:style/Theme.Dialog">
```

Page 45 Fall 2012
Views (Input Controls/widgets)

- **TextView**
  - Label
- **EditText**
  - Editable text
- **Button**
  - Standard push button
- **CheckBox**
  - Standard checkbox
- **RadioButton**
  - Standard radio button
- **ListView**
  - View group that manages a group of Views.
- **Spinner**
  - TextView associated with ListView. Let's you select an item from a list.
Other Views and Widgets

- Widget and Other View
  

- Package android.widget
  

- view - API Demos (Sample Code)
  
Event Listener (Input Events)

• An interface, in the View class, that contains a single callback method

• Called by the Android framework when the View is triggered

• Multiple EventListeners (Interfaces):
  • View.OnClickListener - onClick()
  • View.OnLongClickListener - onLongClick()
  • View.OnFocusChangeListener - onFocusChange()
  • View.OnKeyListener - onKey()
  • View.OnTouchListener - onTouch()
  • View.onCreateContextMenuListener - onCreateContextMenu()
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ... 
        Button button = (Button) findViewById(R.id.button_1);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button_1:
            // do action when the button is clicked
            break;
        // handle more buttons in the activity
        }
    }
}
Menus
Menus

• Options Menu
  – Compact menu bottom of the screen

• Context Menu
  – Long lick on an element

• Popup Menu
  – List of items based on the current view
public class MenuExample extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
    }

    // Modify menu items dynamically. Disable/enable menu items.
    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        ...
    }

    // Create your menu here
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ...
    }

    // When menu item is selected
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ...
    }
}
Creating Menus in XML (res/menu/menu.xml)

Creating a Menu Resource

```xml
<?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" />
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

Inflating a Menu Resource

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```
Responding to Menu Selection

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.new_game:
        newGame();
        return true;
    case R.id.help:
        showHelp();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);
    MenuItem menuItem1 = menu.findItem(R.id.new_game);
    menuItem1.setEnabled(false);
    MenuItem menuItem2 = menu.getItem(0);
    menuItem2.setEnabled(false);
    return true;
}
Submenu

• A sub menu can be added within any menu

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:icon="@drawable/file"
          android:title="@string/file">
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```
Creating Menus in Java

public class MenuExample extends Activity {
    // You can increment this for additional menu items
    static final private int MENU_ITEM1 = Menu.FIRST;
    static final private int MENU_ITEM2 = Menu.FIRST + 1;

    // Create your menu here
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);

        // Specify group value to separate Menu Items for batch processing and
        // ordering. NONE = No group
        int groupId = Menu.NONE;

        // Unique identifier for each menu item (used for event handling)
        int menuItemId1 = MENU_ITEM1;
        int menuItemId2 = MENU_ITEM2;

        // Order value. Menu.NONE = I don't care
        int menuItemOrder = Menu.NONE;

        // Menu Text
        int menuItemText1 = R.string.menu_item1;
        int menuItemText2 = R.string.menu_item2;

        menu.add(groupId, menuItemId1, menuItemOrder, menuItemText1);
        menu.add(groupId, menuItemId2, menuItemOrder, menuItemText2);
        return true;
    }
}
Context Menu

• Context Menu = Long press

• Override two methods on a Activity
  – onCreateContextMenu()
  – onContextItemSelected()

• Register the context menu to the view
  – registerForContextMenu(view)

For details on menu, and other resources:

http://developer.android.com/guide/topics/resources/available-resources.html
public class MyContextMenu extends Activity {
    private TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
        ...
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        ...
    }
}
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    tv = new TextView(this);
tv.setText("Context Menu!");

    registerForContextMenu(tv);
    //or
    //tv.setOnCreatContextMenuListener(this);

    setContentView(tv);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    if (v == tv) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.mymenu, menu);
    }
}
@Override
public boolean onContextItemSelected(MenuItem item) {
    super.onContextItemSelected(item);

    switch (item.getItemId()) {
    case R.id.a:
        Toast.makeText(this, "A", 1000).show();
        return true;
    case R.id.b:
        Toast.makeText(this, "B", 1000).show();
        return true;
    }

    return false;
}
User Notification
Notifications

• Toast Notification
  – Brief message

• Status Bar Notification
  – Persistent reminder

• Dialog Notification
  – Activity–related notification
Toast

- Toast.makeText(this, "Reset", 1000).show();
Status Bar Notification

• Adds an icon to the system's status bar
  – Expanded message in the notification window
  – Optional ticker–text message
  – Highly configurable

• If your app is working in the background, give status bar notifications
The Basics

• Usually a Service is launching the status bar notification

• You must use two classes: Notification and NotificationManager

• Instance of Notification should have all the information about the notification (e.g., icon, the expanded message)

• NotificationManager is an Android system that executes and manages all the notifications

• You don't instantiate NotificationManager
String ref = Context.NOTIFICATION_SERVICE;

NotificationManager mNotificationManager = (NotificationManager) getSystemService(ref);
Instantiate the Notification

Notification notification =
new Notification(R.drawable.notify_icon, "Hello",
System.currentTimeMillis());
Define the Notification's expanded message and Intent

// Communication between status bar and this Activity

Intent notificationIntent = new Intent(this, MyContextMenu.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                   notificationIntent, 0);

notification.setLatestEventInfo(this, "My notification", "Hello World!",
                                contentIntent);
mNotificationManager.notify(1, notification);
Status Bar Notification

String ns = NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager) getSystemService(ns);

int icon = R.drawable.icon;
CharSequence tickerText = "Calculator Reseted!";
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);

Intent notificationIntent = new Intent(this, MyActivity.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);

notification.setLatestEventInfo(this, "My Calculator", "The calculator was reseted!", contentIntent);

mNotificationManager.notify(1, notification);

Dialogs

- AlertDialog
- ProgressDialog
- DatePickerDialog
- TimePickerDialog
Example of AlertDialog

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Hello!")
    .setPositiveButton("Ok", new DialogInterface.OnClickListener()
        { public void onClick(DialogInterface dialog, int id) { dialog.cancel(); }
        })
    .setNegativeButton("Cancel", new DialogInterface.OnClickListener()
        { public void onClick(DialogInterface dialog, int id) { dialog.cancel(); }
        });

AlertDialog alert = builder.create();
alert.show();

Example of Progress Dialog

ProgressDialog dialog = ProgressDialog.show(this, "", "Loading. Please wait...");

dialog.show();
Logging
Android Logging Options

- The `Log.e()` method is used to log errors.
- The `Log.w()` method is used to log warnings.
- The `Log.i()` method is used to log informational messages.
- The `Log.d()` method is used to log debug messages.
- The `Log.v()` method is used to log verbose messages.
- The `Log.wtf()` method is used to log terrible failures that should never happen. (“WTF” stands for “What a Terrible Failure!” of course.)

```java
private static final String TAG = "MyApp";
Log.i(TAG, "I am logging something informational!");
```
Debug (Window > Open Perspective > Debug)

```xml
<application android:icon="@drawable/icon" android:label="@string/app_name" android:debuggable="true"/>
```
Tools

• DroidDraw: http://www.droiddraw.org/

• Video: http://marakana.com/forums/android/g
Questions?
Recommended Assignment: Try It Out

• p38: Understanding the Life Cycle of an Activity
• p42: Displaying a Dialogue Window using an Activity
• p50: Displaying the progress of an operation
• p54: Linking Activities with Intents
• p130: Understanding Activity Behavior when Orientation Changes
• p161: Using the Basic Views
• p168: Handling View Events
• p191: Displaying a Long List of Items using the ListView
• p197: Checking which Items are Selected
• p235 Creating the Menu Helper Methods
• p238: Displaying an Option Menu
• p240: Displaying a Context Menu