UI, Continued

CS 2046

Mobile Application Development

Fall 2010

Announcements

- Office hours have started
- HW1 is out, due Monday, 11/1, at 11:59 pm

- Clarifications on HW1:
  - To move where the text appears in a multi-line EditText, use the android:gravity attribute.
  - When submitting, zip up your entire Eclipse project directory and submit the .zip file.

Recap

- Views, ViewGroups
- Defining layout in XML
- Started with event handling

Anonymous Inner Classes

- Generally, this approach looks nicer if:
  - We are only defining/instantiating this class once.
  - The class is ~10 lines or less.

- Regular inner classes for:
  - Classes which only apply to the outer class
  - Classes which are relatively short (~20-40 lines)

- For any longer classes, or classes which stand alone, separate into a new file.

Event Handling

- Can use setOnClickListener when a View should handle a click event.
  - Saw this in lab – launch Dialog with a button

- Other handlers:
  - setOnKeyListener: When a keyboard key is pressed
  - setOnLongClickListener: on click and hold
  - setOnTouchListener: When the view is touched
    - Different from click – could highlight button and push enter

Other Widgets

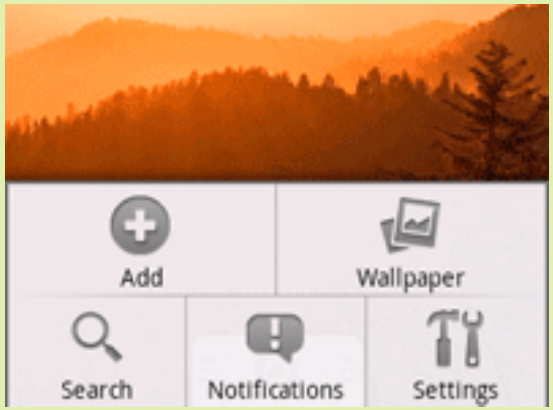
- All of these handlers apply to any View.
- Other widgets define different handlers:
  - CheckBox: setOnCheckedChangeListener
  - DatePicker: init function with argument for OnDateChangedListener
  - More – see Android developer docs for widget you are using.

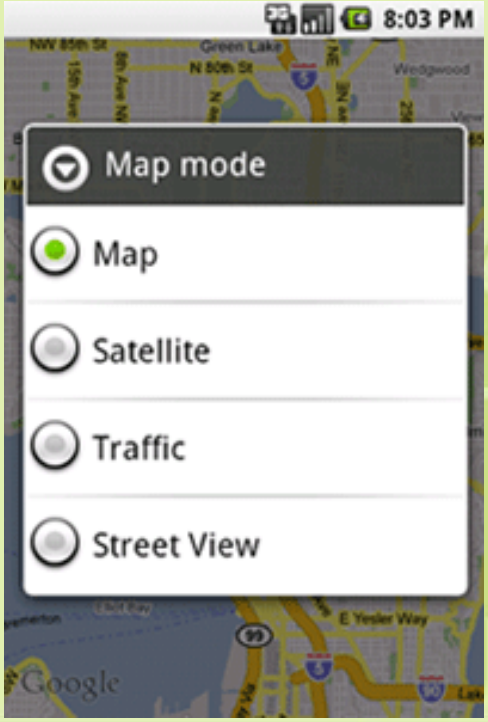
Touch Mode

- Different behavior when user is using touch screen vs. other methods of input.
- Activated when user touches screen
  - Certain views can no longer be “focused on”
    - “focused on” – highlighted, receiving keyboard events
  - These views will simply fire onClick listeners when pressed.
- Deactivated if user uses directional key or scrolls with trackball.

Menus

- Two types:
  - Options menu
    - Press MENU key in an Activity
    - Supports icons, but not checkboxes or radio buttons
    - At most 6 items
      - If >6, 6th icon is “More”, which launches an expanded menu
  - Context menu
    - Appears when long-pressing a View
- Both used in Assignment 1





Defining Menus

- As with anything else that isn't strictly code, can/should be defined in XML.
  - For Assignment 1: Code is okay

- Example:

```
<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/quit"
        android:icon="@drawable/ic_quit"
        android:title="@string/quit" />
</menu>
```

Using XML Menus

- Need to “inflate” XML resource
  - Convert into a Java object
  - Common Android task when converting R.* to it’s corresponding Java object.

- For options menu, here’s an example:

```
public boolean onCreateOptionsMenu(Menu menu) {
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.game_menu, menu);
   return true;
}
```

Options Menu

- Override onCreateOptionsMenu() and onOptionsItemSelected() to display a menu.
  - See assignment 1 for more on this.

- Suppose we want to change the menu.
  - Example: Have a toggled option in the menu.
  - Override onPrepareOptionsMenu() – this gets the Menu that was created in onCreateOptionsMenu(), allows you to add/remove or enable/disable items.

Context Menu

- Register a View to display a context menu with setOnCreateContextMenuListener(…)
  - Usual input – “this”, or your current Activity
  - Must implement onCreateContextMenu() and onContextItemSelected()
  - Again, see assignment 1 for further explanation.

Other Menu Features

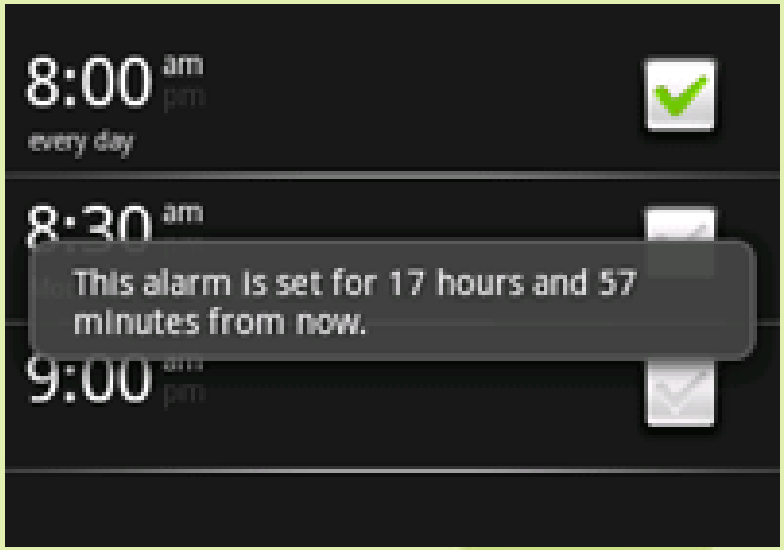
- Submenus
  - Only difference is appearance on screen; still receive events in the parent menu's handler.
- Menu groups
  - Link related items so they can be displayed/hidden, or enabled/disabled together.
  - e.g. with text selected, want to be able to cut and copy.
- Checkable items
- Shortcut keys
- See http://developer.android.com/guide/topics/ui/menus.html

Notifications

- How can we alert the user that something has happened?
- Three cases we'll consider:
  - User is waiting for event, doesn't need to respond.
  - User is waiting for event in a background application, but should be able to respond at any time.
  - User must wait for an event to complete before application can be used again.

Toasts

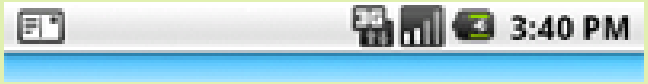
- Case 1: Small notification, no response necessary.
- Best for:
  - Short messages
  - When you are reasonably sure the user is looking at the screen.

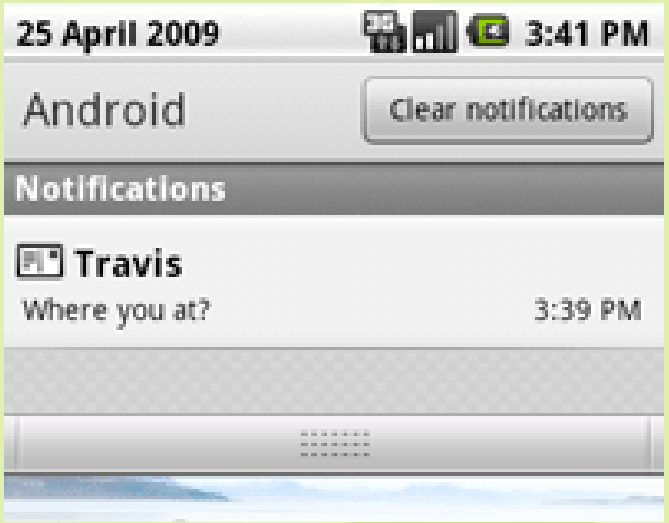



```
Toast.makeText(context, “Message”, Toast.LENGTH_SHORT).show();
```

Status Bar Notifications

- Adds icon to status bar, and item to select for handling event.
  - E.g. missed call, new mail
- Ideal when:
  - Working in background
  - Event has a response (i.e. open email)



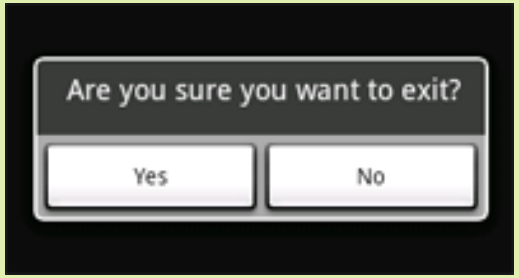


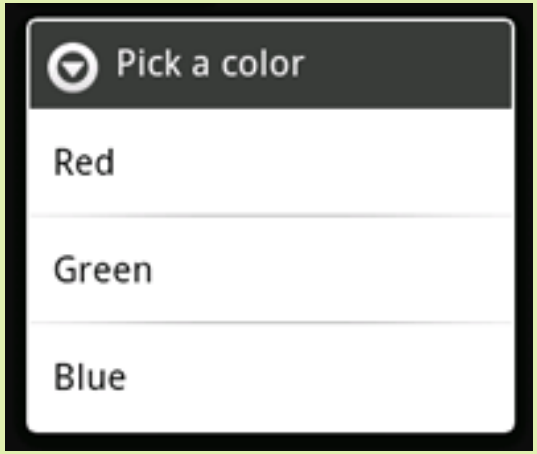
Creating Status Bar Notifications

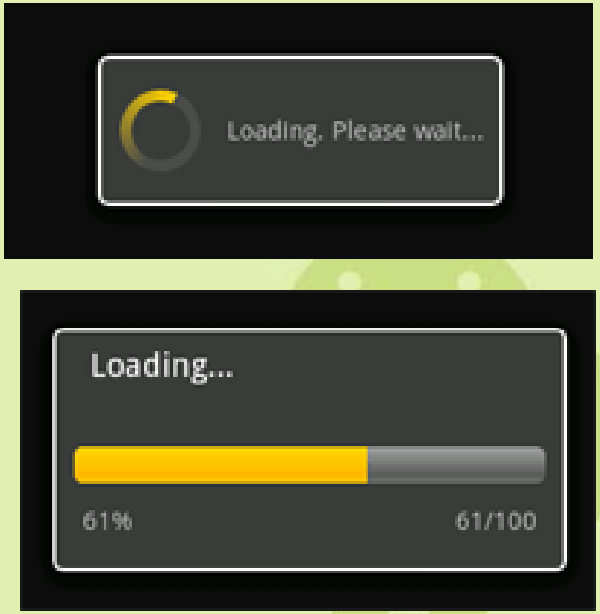
- More complex than Toast
- Use NotificationManager
  - Obtain instance with getSystemService
- Create a notification (icon, text)
- Create an Intent to launch an Activity when the notification is pressed.
  - Also use a PendingIntent object
    - Lets another application launch an Intent as if it was you
- Pass notification to NotificationManager
- See http://developer.android.com/guide/topics/ui/notifiers/notifications.html

Creating Dialogs

- Best for notifying user about something happening in the current application
  - Or, for getting input from the user
- Many different types:







Dialog Types

- Alerts:
  - Use AlertDialog.Builder
    - You did this in lab!
- Progress dialog:
  - Create new ProgressDialog instance, set properties, then call show()
  - Two main types: indeterminate (spinner), or determinate (progress bar)
- Custom dialog:
  - Define XML layout, call dialog.setContentView

Custom Views

- High-level approach:
  - Extend existing View class or subclass
  - Override on* methods
    - i.e. onDraw()/onMeasure(), onKeyDown()
  - Use new method in layout
    - In XML, use full package + class name i.e. <cornell.cs2046.CustomWidget>

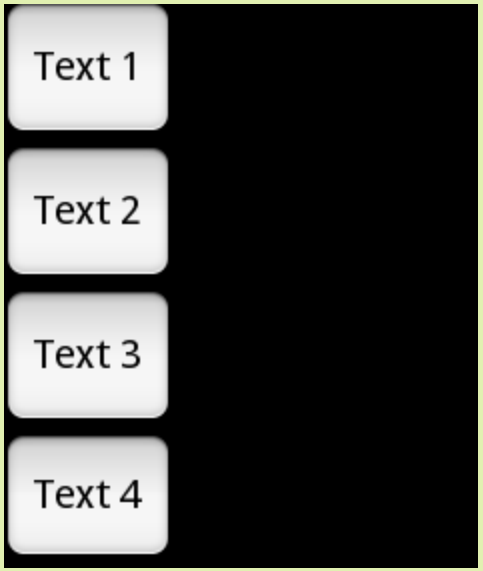
- Not sure what to do?
  - Android is open source – see how common views were created!

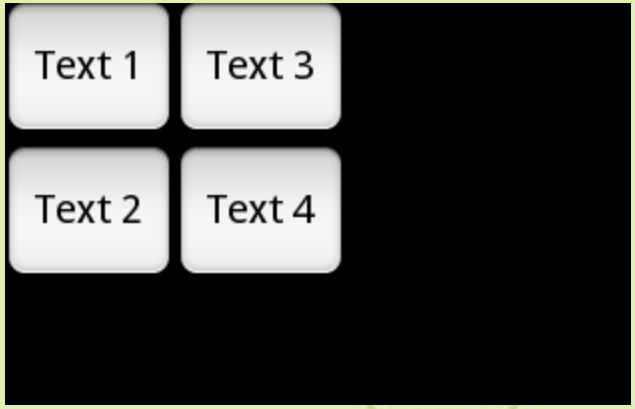
Custom 2D View

- Override onDraw()
  - Input – Canvas object.
  - Can draw any 2D object, text, shape, etc.
- Override onMeasure()
  - Input – restriction on how big the View should be
  - Call setMeasuredDimension to set how big it will actually be
    - Can exceed restriction – in this case, parent View/ViewGroup decides what to do.

Different Screen Configurations

- How can you define different layouts for portrait vs. landscape?





- Answer: Portrait in res/layout, Landscape in res/layout-land

Different Screen Configurations

- Other concerns:
  - Resolution of the screen
  - Density of the screen (dpi)

- Methods of handling:
  - 9-Patch drawables – create scalable graphics
  - Different resources (icons, layouts) for different DPIs
  - Proper size units (dip, or sip = scale-independent pixel)

- Sample which addresses these concerns:
  http://developer.android.com/resources/samples/MultiResolution/index.html

Gestures

- Good replacement for keyboard shortcuts
- Android has built in detectors for these simple gestures:
  - Fling
  - Long press
  - Scroll
  - Tap/Double-Tap
  - Pinch zoom (as of Android 2.2)

Using GestureDetector

- Create a new instance of GestureDetector (or ScaleGestureDetector), passing in a listener.
- In your overridden onTouchEvent, call onTouchEvent for the detector.
- The listener that was passed in will receive any gestures that occur.

Custom Gestures

- Define more custom gestures with Gestures API.
- General approach:
  - Use Gestures Builder to design gestures.
  - Gestures file is saved to /sdcard/gestures
  - In application, use an overlay view to sense gestures.
- Can also have user-defined gestures.

Gesture Sample Code

- In XML layout, <android.gesture.GestureOverlayView> is a blank canvas on which gestures are drawn.
- Can have children – in this case, gestures are drawn over these children.
- Gestures can be stored as resource in res/raw directory, or loaded from a file.

Gesture Sample Code

- In Java:

```
mLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
GestureOverlayView gestures = (GestureOverlayView)
   findViewById(R.id.gestures);
gestures.addOnGesturePerformedListener(this);

public void onGesturePerformed(GestureOverlayView overlay, Gesture
       gesture) {
   ArrayList<Prediction> predictions = mLibrary.recognize(gesture);

   if (predictions.size() > 0) {
       Prediction prediction = predictions.get(0);
       if (prediction.score > 1.0) {
           // Process gesture...
       }
   }
}
```

More on Gestures

- Gestures API was introduced in Android 1.6
  - If attempting karma problem on HW1, you can change the SDK level and build target in Project Properties.

- For more information, see article at: http://developer.android.com/resources/articles/gestures.html

Android UI Guidelines

- Platform guidelines are critical if your application will be on the market.
- UI is important!
  - First impression your user gets of your application
- Guidelines establish a standardized look for applications.

Before releasing an application…

- Make sure icons fit the Android style
  - http://developer.android.com/guide/practices/ui_guidelines/icon_design.html
  - Use stock icons when possible
- Make sure you adhere to Activity design guidelines.
  - http://developer.android.com/guide/practices/ui_guidelines/activity_task_design.html
  - e.g. What does the user expect the back button to do?
- Other guidelines:
  http://developer.android.com/guide/practices/ui_guidelines/index.html