INTERMEDIATE ANDROID

CSCI 4448/5448: OBJECT-ORIENTED ANALYSIS & DESIGN

LECTURE 16 — 03/03/2011

1

Goals of the Lecture

- Dig deeper into the Android Framework

    - Screen Orientation

    - Animation

    - Dialogs

    - Playing Sounds

    - (Simple) Networking

Thursday, March 3, 2011

Android Development Philosophy

- As I learn more about Android development, I'm beginning to understand the Android Development Philosophy

    - "Everything is a Resource"

- or

    - "It's resources all the way down…"

- Many of the steps in Android programming depend on creating resources and then loading them or referencing them (in XML files) at the right time

Thursday, March 3, 2011

Screen Orientation

- People can easily change the orientation by which they hold their mobile devices

  - Mobile apps have to deal with changes in orientation frequently

  - We saw iOS support for multiple orientations in our last lecture

 - Let's see how Android deals with this issue (hint: resources)

Start with Portrait Orientation

- It is natural to start by designing the UI of your main activity in portrait orientation

  - That is the default orientation in the Eclipse plug-in

  - Here's a typical layout for the "main screen" of a game

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/background"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="30dip">
    <LinearLayout
        android:orientation="vertical"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_gravity="center">
        <TextView
            android:text="@string/main_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginBottom="25dip"
            android:textSize="24.5sp" />
        <Button
            android:id="@+id/continue_button"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/continue_label" />
        <Button
            android:id="@+id/new_button"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/new_game_label" />
        <Button
            android:id="@+id/about_button"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/about_label" />
        <Button
            android:id="@+id/exit_button"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/exit_label" />
    </LinearLayout>
</LinearLayout>
```

Thursday, March 3, 2011
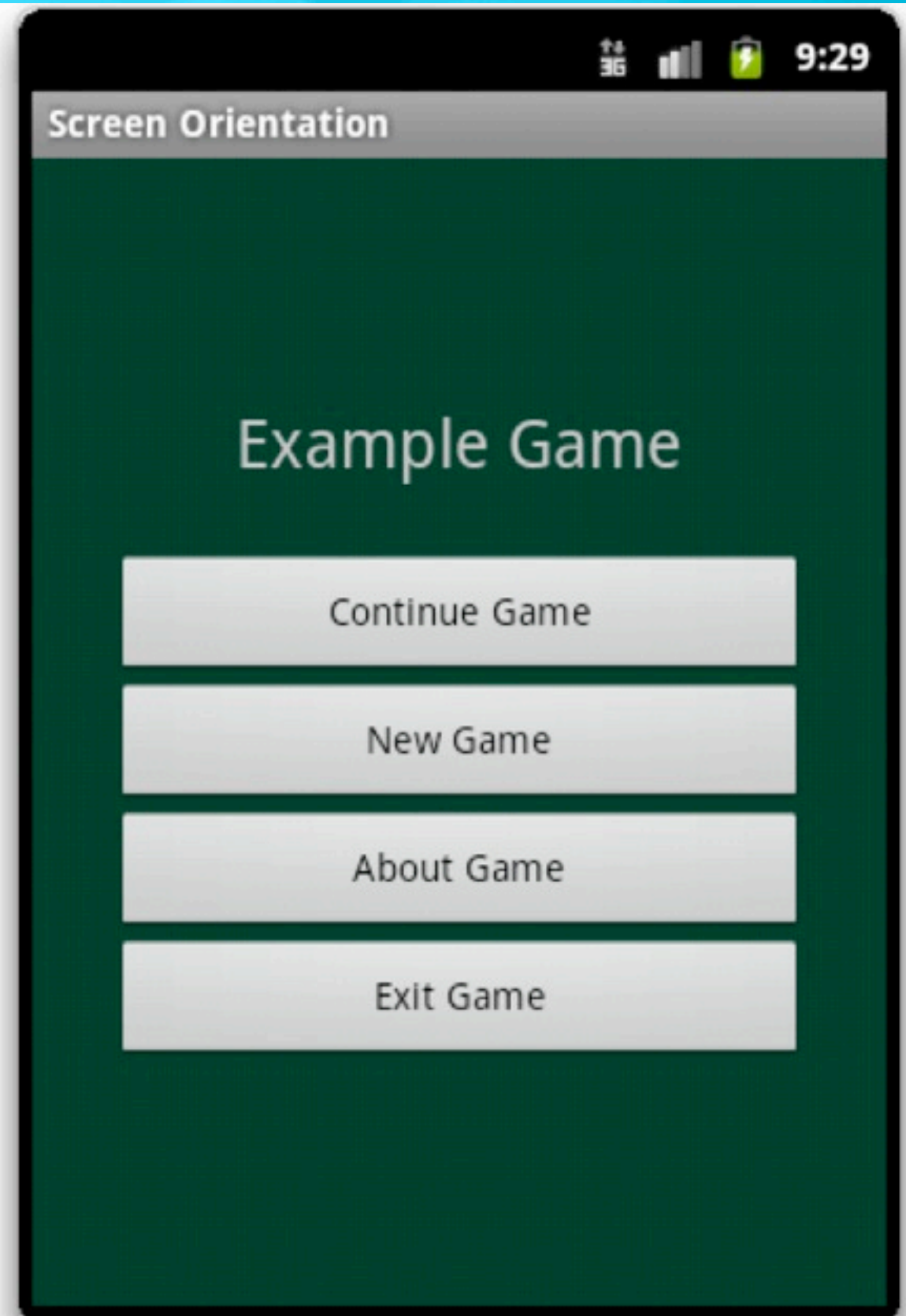
Quick Interjection: Unit Sizes

- Android supports a wide variety of unit sizes for specifying UI layouts; here are all but two

    - px (device pixel), in, mm, pt (1/72nd of an inch)

- All of these have problems creating UIs that work across multiple types of devices

    - Google recommends using resolution-independent units

        - **dp** (or **dip**): density-independent pixels

        - **sp**: scale-independent pixels

- In particular, use **sp** for font sizes and **dip** for everything else

But switch to landscape mode in the emulator (Ctrl+F12) and a problem becomes evident
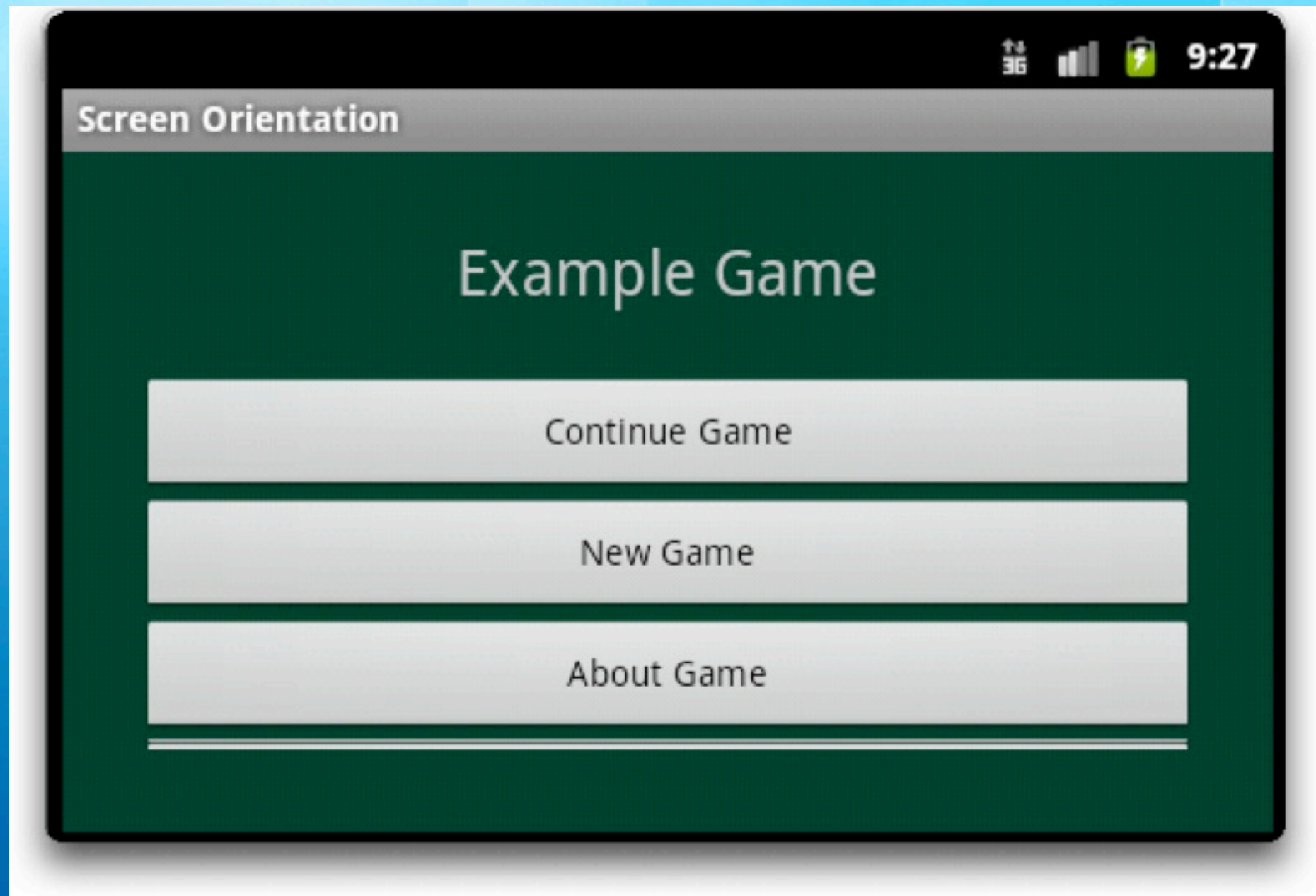
Resources to the Rescue!

- To solve this problem, we create a new main.xml file that has been created specifically for landscape orientation

- This file will live in a new subfolder in the **res** folder of our Android project: res/layout-land/

- This folder is **not** created by default; right click on the **res** folder and select New ⇒ Folder

- Then you can right click on the existing main.xml and select copy and then right click on layout-land and select paste; Finally, you can edit the file for the new orientation

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/background"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:padding="15dip"
    android:orientation="horizontal" >
    <LinearLayout
        android:orientation="vertical"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_gravity="center"
        android:paddingLeft="20dip"
        android:paddingRight="20dip" >
        <TextView
            android:text="@string/main_title"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_gravity="center"
            android:layout_marginBottom="20dip"
            android:textSize="24.5sp" />
        <TableLayout
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_gravity="center"
            android:stretchColumns="*" >
            <TableRow>
                <Button android:id="@+id/continue_button" android:text="@string/continue_label" />
                <Button android:id="@+id/new_button" android:text="@string/new_game_label" />
            </TableRow>
            <TableRow>
                <Button android:id="@+id/about_button" android:text="@string/about_label" />
                <Button android:id="@+id/exit_button" android:text="@string/exit_label" />
            </TableRow>
        </TableLayout>
    </LinearLayout>
</LinearLayout>
```

This layout arranges the buttons into two rows and two columns using a TableLayout
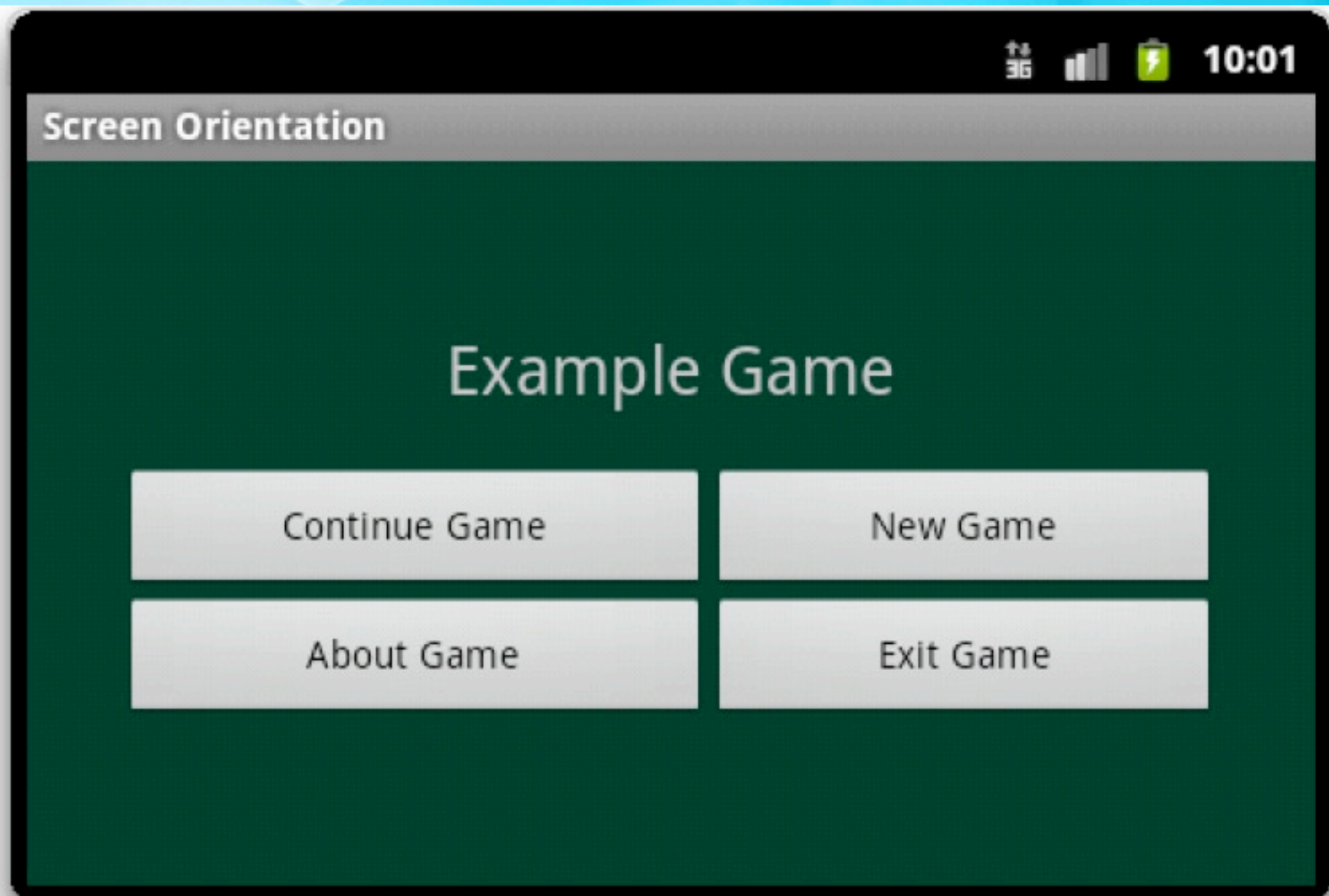
Problem solved. Android automatically switches the layout behind the scenes when the orientation of the device changes.

Types of Layouts?

- **LinearLayout**: Each child view is placed after the previous one in a single row or column

- **RelativeLayout**: Each child view is placed in relation to other views in the layout or relative to its parent's layout

- **FrameLayout**: Each child view is stacked within a frame, relative to the top-left corner. Child views may overlap.

- **TableLayout**: Each child view is a cell in a grid of rows and columns

Specifying the Size of a View

- We've previously discussed the use of resolution-independent measurements for specifying the size of a view

    - These values go in the XML attributes

        - android:layout_width and android:layout_height

- But, you can get more flexibility with

    - fill_parent: the child scales to the size of its parent

    - wrap_content: the parent shrinks to the size of the child

Thursday, March 3, 2011

Animating Views

- Android offers four different ways of performing animation

    - **Support for Animated GIF images**

    - **Frame-by-Frame animation**: developer supplies images and specifies transitions between them

    - **Tweened animation**: simple animation effects that can be programmatically applied to views

    - **OpenGL ES**: advanced 3D drawing, animation, etc.

Thursday, March 3, 2011

Tweened Animation

- Tweened animations are specified (unsurprisingly) via resources

- The basic process involves doing the following in the onCreate () method of the Activity

    - get a handle to the view

    - load the animation resource: such as fade

    - apply it to the view: view.startAnimation(fade)

- Android provides animation support for alpha, rotation, scaling and translating; the first deals with transparency; the third deals with a view's size; the last deals with moving views around

Thursday, March 3, 2011

Our Plan

- We'll apply animations to the buttons defined on the portrait layout of the previous example

- We'll make one fade in, one rotate, one scale, and one that does all three at once!

  - We'll also have each animation happen one after the other

  - In a real application, this would get tedious, but for illustration purposes, it's fine

Thursday, March 3, 2011

The Process (I)

- Step One: Use the New Folder command to create a folder called anim in the res folder of our project

- Step Two: Create a new Android XML File in the anim subfolder, call it fade.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <alpha>
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:duration="1000"
    </alpha>
</set>
```

17

Thursday, March 3, 2011

The Process (II)

- Step 3: Add the following code to the Main activity's onCreate() method

```
Button continue_button = (Button) findViewById(R.id.continue_button);

Animation fade = AnimationUtils.loadAnimation(this, R.anim.fade);

continue_button.startAnimation(fade);
```

- You will need these import statements

```
import android.view.animation.Animation;

import android.view.animation.AnimationUtils;

import android.widget.Button;
```

18

Thursday, March 3, 2011

The Process (IV)

- There are no additional steps… just run the program!

    - Demo of "Fun With Animation"

- As you saw from the code, we used the attribute

    - android:startOffset

- to control when particular animations start

- So, Android makes it straightforward to perform simple animations within Android apps

Getting input from the user

- Android provides several types of default dialog boxes

  - and provides a way to create custom dialogs as well

- The dialog types

  - Dialog (the base class for all dialogs; you subclass this class to create custom dialogs)

  - AlertDialog (a dialog with 1-3 buttons)

  - DatePicker and TimePicker

  - ProgressDialog (both determinate and indeterminate)

© Kenneth M. Anderson, 2011

20

Dialog Life Cycle (I)

- Each activity manages the life cycle of the dialog boxes it displays to its users
  - It calls showDialog()to display a dialog
    - That dialog gets added to its dialog window cache
  - It calls dismissDialog() to remove a dialog window and keep it in the cache; subsequent display is faster
  - It calls removeDialog() to remove the dialog from the cache

Dialog Life Cycle (II)

- Each dialog has an associated id; you pass that id to showDialog()

    - This causes the method onCreateDialog() to be called with that id. You then use a switch statement to create the appropriate dialog based on the id

        - onCreateDialog() is typically called once; thereafter the dialog is retrieved from the cache

    - The next method called is onPrepareDialog(); this method is called whenever the dialog is about to be shown

Example

- Let's create an app that shows how to use

  - AlertDialog

  - DatePicker

  - TimePicker

- We'll see the use of a ProgressDialog a little bit later

- Demo of "Fun With Dialogs"

Discussion

- Code looks more complex than it actually is

    - In the onCreateDialog() method, we simultaneously create the dialogs that we need PLUS the methods that act as the dialog's event handlers

    - In the onPrepareDialog() method, we either reuse the previously set value (stored in attributes) or we set the dialog to a default value (current day and current time)

Thursday, March 3, 2011

Playing Sounds

- Android makes it very easy to play sounds

  - You copy supported sound files to res/raw

  - You create an instance of MediaPlayer

    - Such as MediaPlayer mp;

    - When you want the sound to play, you call mp.start and pass in the resource id of the sound you want

    - When you want the sound to stop, you call mp.stop

- Demo of SoundPlayer

Thursday, March 3, 2011

Networking (I)

- Mobile apps will often need to access a web service or web page to retrieve information that it then displays to its user

- In Android, accessing network resources must always occur in a thread that is separate from the GUI thread

  - Otherwise, the GUI thread can be blocked waiting for a remote server to respond and the user will think that the application has crashed

Networking (II)

- There is nothing magic about Android's networking

    - Your program can use any of Java's IO packages to access the internet

    - The trick is that you must run that code in a thread

- Android offers two ways of running tasks asynchronously

    - AsyncTask and Thread/Handler

    - The latter requires the developer to do all the work, so we will look at the former

Thursday, March 3, 2011

Networking (III)

- AsyncTask is an abstract class that makes it straightforward to run a task in the background that also updates the GUI

- To use, you create a subclass of AsyncTask and override the following methods

  - **onPreExecute()** - runs on the GUI thread before the background process is started

  - **doInBackground()** - contains the code for the background process

Networking (IV)

- To use, you create a subclass of AsyncTask and override

  - **onProgressUpdate()** - runs on the GUI thread and contains information passed from the background thread; to do this, the background thread, passes information to a method called **publishProgress()**

  - **onPostExecute()** - runs on the GUI thread, once the background process is done

Networking (V)

- So, for a standard hit on a web service, you would

  - set up a progress bar in onPreExecute()

  - call the web service in doInBackground()

    - when you receive a result, loop over the contents and call publishProgress() with info

  - in onProgressUpdate() update the progress bar or update the GUI with information from the web service or both

  - make the progress bar go away in onPostExecute()

Thursday, March 3, 2011

Java Feature: varargs

- The AsyncTask class makes use of Java's version of sending a method a variable number of arguments

- The syntax looks like this

  - public void process(String… args);

- Inside the method, args acts just like a Java array but defining it this way allows you to pass in any number of strings to process, be it as an array or as individual string arguments

Thursday, March 3, 2011

The progress indicator

- We'll create an instance of ProgressDialog to show let our user know that data is being downloaded and processed

  - Since we don't know how long the download will take, we will use an indeterminate progress indicator

    - This type of progress bar displays a spinning image to let the user know that the program hasn't crashed

Our Example (I)

7108558 Total Tweets
1302111 Total Users

back (0) | forward (2)

<< first < prev 1 **2** **3** **4** **5** next > last >>

Term	Doc Count
rt	3284715
http	3000462
libya	2622876
gaddafi	1081669
bit.ly	1022043
egypt	770192
bahrain	601764
feb17	537366
from	472701
i	465658
people	363758
christchurch	342927
revolution	328845
jordan	325222
jan25	319791
you	312326
earthquake	302508

<< first < prev 1 **2** **3** **4** **5** next > last >>

Tweet Source	Tweet Count
web	2972281
Twitter for iPhone	609906
TweetDeck	543393
twitterfeed	520263
Twitter for BlackBerry®	395979
Echofon	149746
HootSuite	133278
Tweet Button	131194
ÜberSocial	122692
Mobile Web	117243
Twitter for Android	85899
dlvr.it	85631
Twitter for iPad	83269
Google	62571
Twitter for Mac	52971
Snaptu	45048
Facebook	41901

Home
Events
Searches
Filters
System
Logout

EPIC
Colorado

Thursday, March 3, 2011

Filters

Our Example (II)

Create Filter | View Disabled

ID	Creation Date ▾	Query	Enabled	Actions
1	02/17/2011	algeria	☑	Edit
2	02/17/2011	algérie	☑	Edit
3	02/17/2011	algerie	☑	Edit
4	02/17/2011	bouteflika	☑	Edit
5	02/17/2011	dz2011	☑	Edit
6	02/17/2011	12feb	☑	Edit
7	02/17/2011	12fevrier	☑	Edit
8	02/17/2011	12fev	☑	Edit
9	02/17/2011	12fév	☑	Edit
10	02/17/2011	feb12	☑	Edit
11	02/17/2011	fev12	☑	Edit
12	02/17/2011	fév12	☑	Edit
13	02/17/2011	bahrain	☑	Edit
14	02/17/2011	14feb	☑	Edit
15	02/17/2011	feb14m	☑	Edit
16	02/17/2011	bh14feb	☑	Edit
17	02/17/2011	feb11	☑	Edit
18	02/17/2011	11feb	☑	Edit
19	02/17/2011	tahrir	☑	Edit
20	02/17/2011	jordan	☑	Edit

Home
Events
Searches
Filters
System
Logout

EPIC
Colorado

34

Demonstration

- Let's write a simple Android client that uses AsyncTask to hit the web services API of Project EPIC to get the current set of filters

    - We will hit a URL that returns a list of filters in JSON format

    - We'll parse the JSON to get the filter names

    - We'll display the filters in a list

    - We'll demonstrate the use of AsyncTask along the way

    - Note: must set android.permission.INTERNET to access the network

Discussion

- Straightforward example

    - AsyncTask works as advertised

        - creating, displaying, and dismissing progress dialog was a snap

        - very easy to send results from background thread to GUI thread

    - Makes use of some advanced Java constructs to allow a private class to access attributes and methods of its surrounding class

Thursday, March 3, 2011

Wrapping Up

- Learned more about the Android framework

    - How to handle multiple orientations

    - How to handle simple animations

    - How to handle simple dialogs

    - How to play sounds

    - How to handle a simple network request (with progress bars!)

Thursday, March 3, 2011

Coming Up Next

- Homework 5: Start of Semester Project

    - will be released later today

- Lecture 17: Intermediate iOS

- Lecture 18: Review of Midterm

- Homework 5 due next Friday

Thursday, March 3, 2011