Android Programming
Lecture 18:
Menus
Sensors
11/11/2011
Simple Menu Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@id/item1" android:title="Red"></item>
    <item android:id="@id/item2" android:title="Green"></item>
    <item android:id="@id/item3" android:title="Blue"></item>
</menu>
```

```java
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.threeitems, menu);
    // return true if want to display, false if not return true;
}

public boolean onOptionsItemSelected (MenuItem item) {
    super.onOptionsItemSelected(item);
    if (item.getItemId() == R.id.item1) {
        textView.setText("Item 1 selected");
    }
    else if (item.getItemId() == R.id.item2) {
        textView.setText("Item 2 Selected");
    }
    else if (item.getItemId() == R.id.item3) {
        textView.setText("Item 3 Selected");
    }
    textView.setText(textView.getText() +  "\nColor: " + item.getTitle());

    // return true if want to indicate completely handled
    return true;
}
```
Submenu Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1" android:title="Red"></item>
    <item android:id="@+id/item2" android:title="Green"></item>
    <menu>
        <item android:id="@+id/item2a" android:title="Green_SubA"></item>
        <item android:id="@+id/item2b" android:title="Green_SubB"></item>
    </menu>
</menu>
```
Sensors and Actuators
Sensors

• Sensors provide information about the device and its environment
  – Will ignore camera and microphone for now

• Sensors types provided on most Android phones:
  – Accelerometer: Acceleration along three axes
  – Gyroscope/Orientation: Three axis orientation (heading)
  – Light: Ambient light sensor
    • Setting appropriate screen brightness
  – Magnetic field: Three axis magnetic field
    • Car navigation system startup
  – Pressure: Pressure exerted on device
  – Proximity: Distance between device and target (usually electrical field)
    • Turn off screen when near ear so don’t accidentally press buttons
  – Temperature: Some meaningful temperature (ambient, battery, ...)
  – Battery level?
Actuators

• Actuators provide feedback to the user

• Beyond Views we can program, we can access:
  – Lights on the device
  – Notification bar
  – Vibration/force feedback
Sensors: Sensor Manager

- Android provides an interface object for you to work through to request sensor information
  - `SensorManager`
  - There may be more than one physical sensor that can provide a particular type of information

- User of SensorManager parallels LocationManager usage
  - You as a programmer request LocationManager to provide updates
  - LocationManager actually talks to the GPS/Cell Tower/Wireless location sources to get you the data

- Getting access to SensorManager:
  ```java
  SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
  ```
Sensors: What Is Available On My Phone?

• We can request the SensorManager to provide us a list of all sensors available on a phone:

  $$\text{List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);}$$

• Or sensors of a given type (example type list on page 458 of book)

  $$\text{List<Sensor> lightSensors = sensorManager.getSensorList(Sensor.TYPE_LIGHT);}$$
Sensors: Listing Sensors Example

• What is available in the Emulator?

• What is available on my Motorola Droid?

An accelerometer?
Sensors: Registering To Listen For Events

• We have to tell the SensorManager which sensor we want to listen for
  – Again, much like LocationManager ("I want to listen to GPS events")

// get a handle to sensor of interest
Sensor sensor =
sensorManager.getDefaultSensor(Sensor.TYPE_??????);
// request updates
sensorManager.registerListener(thelistener, theSensor,
updateRate)
// ...do work...
// (eventually) unregister from updates when done
sensorManager.unregisterListener(thelistener);

Everything underlined we will need to fill in specific to our app and the sensor of interest to us
Sensors: Registering To Listen For Events

// request updates
sensorManager.registerListener(thelistener, theSensor, updateRate)

Who is listening to what, and how fast are they listening?

theListener: usually your class (this)
theSensor: from previous line of code Sensor sensor =
sensorManager.getDefaultSensor(Sensor.TYPE_??????);
updateRate: one of
  SensorManager.SENSOR_DELAY_FASTEST
  SensorManager.SENSOR_DELAY_GAME
  SensorManager.SENSOR_DELAY_NORMAL
  SensorManager.SENSOR_DELAY_UI

See example of how to pull this all together!
Sensors: How to Handle An Update From A Sensor

• Employ the Listener pattern as we have already seen for Views and Location information

Two functions that are called:

1) Reporting a change in accuracy
2) Reporting a change in value
Sensors: SensorEvents

- A *SensorEvent* contains the following information:

  ```java
  public int accuracy
  The accuracy of this event. See SensorManager for details.
  
  public Sensor sensor
  The sensor that generated this event. See SensorManager for details.
  
  public long timestamp
  The time in nanosecond at which the event happened.
  
  public final float[] values
  The length and contents of the values array depends on which sensor type is being monitored (see also SensorEvent for a definition of the coordinate system used).
  ```

- Note these are all public variables!
  - Assuming you have a SensorEvent variable called `event`, you don’t call `event.getAccuracy()`, you just do `event.accuracy`

- values array will be filled in as appropriate:
  - Some sensors only use values[0]
  - Others employ more than one spot
public class SensorTestActivity extends Activity implements SensorEventListener {

    TextView textView;
    SensorManager sensorManager;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textView = (TextView)findViewById(R.id.text_view);

        SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
        for (int i = 0; i < allSensors.size(); i++){
            Sensor sensor = allSensors.get(i);
            String sensorDescription = sensor.getType() + " " + sensor.getName() + " " + sensor.getVendor();
            textView.setText(textView.getText() + "\n" + sensorDescription);
        }

        textView.setText("Starting listening to light sensor...");
        Sensor lightSensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
        sensorManager.registerListener(this, lightSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }

    public void onDestroy() {
        sensorManager.unregisterListener(this);
    }

    public void onSensorChanged(SensorEvent sensorEvent) {
        float distance = sensorEvent.values[0];
        textView.setText(textView.getText() + "\n" + distance + " lux");
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
}
Sensors: Example: Accelerometer

Imagine device sitting on table

X-axis: lateral movement (left/right), values[0]
  + values → moving to its right
  - values → moving to its left

Y-axis: longitudinal movement (forward/back), values[1]
  + values → moving phone in direction of top of phone
  - values → moving phone in direction of bottom of phone

Z-axis: vertical movement (up/down), values[2]
  Always includes force of gravity (9.81 m/s^2)
  Values > 9.81 → moving away from earth
  Values < 9.81 → moving towards earth (freefall)

There are built-in constants in the SensorManager class you can use to normalize some of these (including the gravity of the Death Star?!?)
Sensors: Example: Accelerometer

• Attempt to detect being picked up or falling
• Readings from accelerometer come quickly
• Average last ten readings, compare to 9.81 (+/- a little noise)
Sensors: Example: Accelerometer

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    textView = (TextView)findViewById(R.id.text_view);

    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    textView.setText("Starting listening to accelerometer");
    Sensor accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    sensorManager.registerListener(this, accelerometer, sensorManager.SENSOR_DELAY_NORMAL);
    lastTenVertical = new ArrayList<Double>();
}

public void onSensorChanged(SensorEvent sensorEvent) {
    textView.setText("\n\nLateral (l-to-r): " + sensorEvent.values[0]);
    textView.setText(textView.getText() + \n"\nLongitudinal (f-to-b): " + sensorEvent.values[1]);
    textView.setText(textView.getText() + \n"\nVertical (u-to-d): " + sensorEvent.values[2]);
    textView.setText(textView.getText() + \n"\n");

    if (lastTenVertical.size() < 10) { lastTenVertical.add(new Double(sensorEvent.values[2])); }
    else { lastTenVertical.remove(0);
    lastTenVertical.add(new Double(sensorEvent.values[2]));

    if (lastTenVertical.size() == 10) {
        double average = 0;
        for (int i = 0; i < 10; i++) { average = average + lastTenVertical.get(i).doubleValue();
        average = average / 10;

        if (average < 8.5) {
            sensorManager.unregisterListener(this);
            textView.setText("put me down! " + sensorEvent.values[2]);
        }
        else if (average > 10.9) {
            sensorManager.unregisterListener(this);
            textView.setText("catch me! " + sensorEvent.values[2]);
        }
    }
```
Actuators

• Actuators provide feedback to the user

• Beyond Views in Activities we can program, we can access:
  – Status bar
    • Icon
    • Message
  – Extended status bar
  – Lights on the device
  – Vibration/force feedback
  – Ringing

Can employ all through ‘Notifications’ framework as well as some independently
Actuators

Status bar

Extended status bar

Hardware lights here
Droid has one, w/ multiple colors
Actuators

• Employ NotificationManager
  – Like LocationManager, SensorManager

NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
Actuators: Status Bar

- Notification status bar employs:
  - Icon to be drawn
  - Ticker text to be shown
  - Time of notification (for appropriate ordering) in milliseconds

- Extended status bar employs:
  - Application context
  - Title
  - Expanded ticker text
  - A PendingIntent which can be launched if notification pressed (can set Intent in PendingIntent as null if don’t want to launch anything)

- To use:
  - Create notification
  - Add additional information for extended status bar
  - Send to notification manager with an int reference id
  - Eventually, cancel notification using same reference id
Actuators: Status Bar

• Create notification:
  – `Notification notification = new Notification(int R.drawable.iconReference, String text, long timeInMilliseconds);`

• Add extended status bar information:
  – `notification.setLatestEventInfo(Context context, String expandedTitle, String expandedText, PendingIntent pendingIntent)`

• Send to notification manager
  – `notificationManager.notify(int notificationReferenceID, Notification notification)`

• Eventually, cancel notification
  – `notificationManager.cancel(int notificationReferenceId)`
public class NotificationsTestActivity extends Activity implements View.OnClickListener {

    NotificationManager notificationManager;
    Button button;
    boolean pressed;
    int BUTTON_NOTIFICATION_ID = 1;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);

        notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        pressed = false;
    }

    public void onClick(View view) {
        if (view.getId() == R.id.button) {
            if (pressed == false) {
                Notification notification = new Notification(R.drawable.star,
                        "Button Pressed!", System.currentTimeMillis());

                notification.setLatestEventInfo(getApplicationContext(), "The button was pressed!",
                        "Button Status Update", PendingIntent.getActivity(getApplicationContext(), 0, null, 0));
                notificationManager.notify(BUTTON_NOTIFICATION_ID, notification);
                pressed = true;
                button.setText("Press to cancel notification");
            } else {
                notificationManager.cancel(BUTTON_NOTIFICATION_ID);
                pressed = false;
                button.setText("Press to get a notification");
            }
        }
    }
}
Actuators: Notification
Bells and Whistles

• Notification bells and whistles, such as lights, sounds, and vibration can be associated with a Notification
  – Build a Notification using approaches on previous slides, then associate these extra features with it
Actuators: Vibrate

• Requires permission to vibrate
  – <uses-permission
      android:name="android.permission.VIBRATE"/>
• Define an array of longs which contains times to vibrate off/on/off/on.... (in milliseconds)
• Set notification.vibrate variable to the array specified above
Actuators: Sounds

• Any sound can be associated with a Notification
  – Set \textit{notification.sound} variable to a URI of location on phone that points to sound
  – Can access default ringtones through \texttt{RingtoneManager} class
    • Actually, has separate defaults for ALARM, NOTIFICATION, and RING
Actuators: Lights

• Set color of light: (can be constructed from RGB)
  – `notification.ledARGB=Color.XXX;`

• Set Off length in milliseconds
  – `notification.ledOffMS = time;`

• Set On length in milliseconds
  – `notification.ledOnMS = time;`

• Set FLAG_SHOW_LIGHTS
  – `notification.flags = notification.flags | Notification.FLAG_SHOW_LIGHTS;`

| is bitwise OR; continuously on: ledOnMS = 1, ledOffMS = 0; set off: ledOnMS = 0, ledOffMS = 0