Week 4 Workshop Activity

Intents. Accessing the Sensors of the Device.

In this workshop you will build two applications. The first one is an exercise about intents in which you will start an activity to get a result through an explicit intent, and also start an activity through an implicit intent (viewing a webpage)
The second activity will demonstrate how we can access the sensors of the device.
Code is provided only for the sections in which we introduce new concepts – for the other code, refer to what you done in previous workshops and sample code from lectures.

Objectives of Workshop 4:

1. Use Android intents
2. Learn how to read the various sensors on the device
3. Create event handlers for the accelerometer sensor

Part 1: Exercise Using Intents

The key concept of this first exercise is that of Intent, which allows you to send requests to the Android system to start a new Activity.
We will use an Intent to launch a second Activity and then accept the result of that that Intent to launch a web page.
This exercise is based on the Lecture 4 and the corresponding readings:
http://developer.android.com/training/basics/intents/index.html

Main Activity

The screenshot below illustrates the MainActivity for this exercise. It contains one button:

To Do:

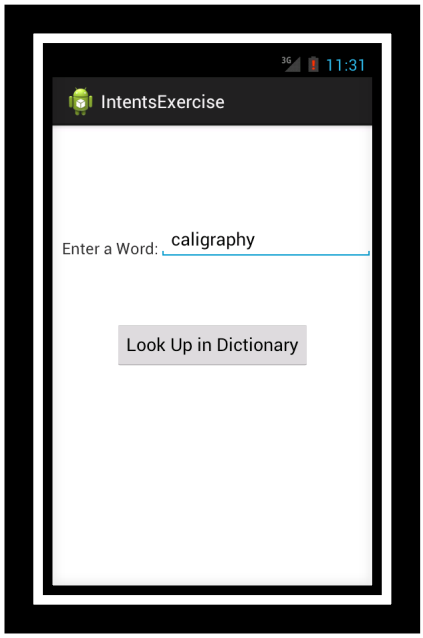
- Create a button in your main activity.
- When the user clicks the button, your app will use an explicit Intent to launch your WordEntryActivity.

```
@Override
public void onClick(View v) {
    Intent i = new Intent(MainActivity.this, WordEntry.class);
    startActivityForResult(i, REQUEST_WORD_ENTRY);
```

REQUEST_WORD_ENTRY is an integer, representing the REQUEST_CODE corresponding to the second activity (WordEntryActivity), and on the basis of which we will retrieve the result (the word entered by the user) from the WordEntryActivity.

WordEntry Activity

The second activity allows your user to enter a word into an EditText box, and return that information to the calling Activity:



To Do:

- Create one EditText box for the word that the user will enter.
- You should also create a label for this box.
- Create a button, which reports the result to the calling activity, which is your MainActivity.

- When the user clicks the button, your activity needs to read the entry from the text box, and return the entered text to the calling activity. It should do that by creating a new Intent, using putExtra on that intent to store entered word, and calling setResult() to set RESULT OK.

```
@Override
public void onClick(View v) {
    word =  wordEntryEditText.getText().toString();

    //verify that the word is captured
    Toast.makeText(this, word,Toast.LENGTH_SHORT).show();

    Intent i = new Intent();
    i.putExtra("WORD", word);
    setResult(RESULT_OK, i);
    finish();
}
```

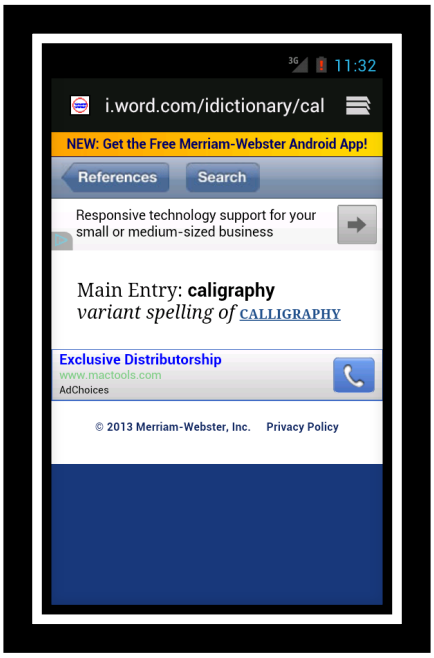
- To return control to the first activity, you need to call finish(). Both setResult() and finish() are defined on the Activity class, so your WordEntryActivity can just call these methods.

Reading off the returned results

The final task is to handle the word that the WordEntry activity returned.

- Implement the onActivityResult() method. If it's called with the appropriate resultCode and requestCode it should start an implicit intent that opens the following URI: http://www.merriam-webster.com/dictionary/[word] (where 'word' is the ext entered by the user in the EditText):

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode==REQUEST_WORD_ENTRY) //check that we're processing the response from WordEntry
    {
        if(resultCode==RESULT_OK) //make sure the request was successful
        {
            if (data.hasExtra("WORD"))
            {
                String word_entered = data.getExtras().getString("WORD");
                Uri webpage = Uri.parse("http://www.merriam-webster.com/dictionary/"+ word_entered);
                Intent webIntent = new Intent(Intent.ACTION_VIEW,webpage);
                startActivity(webIntent);
            }
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```



Part 2: Accessing the Sensors of the Device

The goal of this exercise is to display data from the device's sensors to the user.

Implement a graphical user interface that will allow the user to see what sensors are available on the device, and also to see detailed information for sensors. To start, we will display a list of all sensors in a TextView and then display detailed information for the acceleration sensor (the three values of the acceleration, along x, y, z) – see screenshot below:

For this exercise, you'll need to get that data from the device's sensors. A number of Android classes will allow you to read this data, including:

- **SensorManager** - manages the various sensors of the device.
- **Sensors** - a software representation of the sensor hardware. You get references to specific instances of Sensors from the SensorManager.
- **SensorEventListener** - you must implement this interface to receive events from the sensors.
- **SensorEvents** - represents a single message from a single sensor. The documentation of the SensorEvent has an excellent explanation of the format that each sensor sends its values in.

The onCreate method of the activity gets a reference to the SensorManager, where all sensor-related functions take place. The onCreate method also establishes references to the GUI components that you'll need to update with sensor data values.

```
//get reference to sensor and attach a listener - (acquire sensors late - release early)

mySensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

//get reference to the accelerometer sensor
myAccelerometer = mySensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

The onResume() method uses the reference to the SensorManager to register for sensor updates from the registerListener method:

- The first parameter is an instance of a class that implements the SensorListener interface ('this' as our activity class implements Sensor EventListener interface
- The second parameter refers to the desired sensor. In this case, the application is requesting data from SENSOR_ACCELEROMETER.
- The third parameter is a hint for the system to indicate how quickly the application requires updates to the sensor values.

```
@Override
protected void onResume() {
    super.onResume();
  //listener for the accelerometer sensor
  mySensorManager.registerListener(this, myAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
}
```

When the application (activity) is paused, you want to unregister the listener so you no longer receive sensor updates. This is accomplished with the unregisterListener method of the SensorManager. The only parameter is the instance of the SensorListener. Unregistering event listeners helps the device conserve battery power.

```
@Override
protected void onPause() {
    //unregister listener - release the sensor
     mySensorManager.unregisterListener(this);
    super.onPause();
}
```

A SensorListener must implement the two methods onSensorChange and onAccuracyChanged. This example application is really not concerned with the accuracy of the sensors, but rather the current X, Y, and Z values of the acceleration sensor:

```
@Override
public void onSensorChanged(SensorEvent event)
{
    int type = event.sensor.getType();

    if (type == Sensor.TYPE_ACCELEROMETER)
    {
        accel_vals =event.values;
        accelXEditText.setText("AccelX: " + accel_vals[0]);
        accelYEditText.setText("AccelY: " + accel_vals[1]);
        accelZEditText.setText("AccelZ: " + accel_vals[2]);
    }

}
```

To display the the list of device sensors in a TextView:

```
List<Sensor> mList = mySensorManager.getSensorList(Sensor.TYPE_ALL);

for (int i = 1; i < mList.size(); i++)
{
    sensorsTextView.append("\n" + mList.get(i).getName());
}
```

This exercise has walked you through the basics of acquiring and reading values from the sensors of a device.