A few notes on Android

Getting Started

To get started, create a new project in Android Studio and accept all the defaults, making sure that the minimum API is set to at least 16. At the list of templates, choose `Empty Activity`. Android Studio will generate two files for you: `activity_main.xml` and `MainActivity.java`. The XML file is sort of like a style file that determines what things will look like. The Java file is the code that determines the actions that are taken when things happen. Initially, the app generated by Android Studio is a Hello World app.

Go over to the XML file. There are two tabs here at the bottom – `Design` and `Text`. `Design` is a drag-and-drop editor where you can add things like buttons to your app. `Text` is where you can manually edit the XML layout code. I like to start by modifying the XML file a little. To keep things simple, I use a `LinearLayout` instead of the default layout generated by Android Studio. Type the following into the file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    >
</LinearLayout>
```

For `layout_width` and `layout_height`, the two common choices are `match_parent` and `wrap_content`. The former makes things as big as its parent object (in this case taking up the whole screen). The latter makes an object that is just large enough to hold its content. For instance, if you use `match_parent` for a button's width and `wrap_content` for its height, then the button will be as wide as the screen (or its parent object) and just tall enough to hold its text and a little space around it. The `android:orientation` option here specifies that the linear layout is vertical so that each new item that is added to it will appear vertically below the previous one. The `android:gravity` options here centers things horizontally and vertically.

There are a lot of different layouts, but when you are just starting out, it's probably simplest to just use `LinearLayout`. Now that the layout is set, go back to the `Design` tab. There you can drag and drop a `TextView` and a `Button` onto the screen. Android Studio will automatically generate XML for these. Go over to the `Text` tab to see the XML. It should look something like this (except for one line that I've added):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    >

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click me"
        android:onClick="handleClick" />
```

```
        />

    </LinearLayout>
```

Android automatically generates an identifier for the two widgets. This identifier is important for referencing these widgets from Java. The `android:text` lines specify the text that's in the the `TextView` and in the `Button`. The last line of the button's XML, containing `android:onClick` is one that I manually added. That line specifies the name of a Java method to be called when the button is clicked.

Now let's go over to the Java code. There will be a package statement at the start along with a few imports, followed by this code:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

The `onCreate()` method is called when your app is started. Any code that you want to run on startup can go in here; just make sure that it comes after the two lines that are already here. Modify the code so it looks like this:

```
public class MainActivity extends AppCompatActivity {

    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textView = (TextView) findViewById(R.id.textView);
    }

    public void handleClick(View v) {
        textView.setText("You clicked the button!");
    }
}
```

As you type things in, Android Studio will pop up a message to help you import the necessary libraries. Hit Alt+Enter (or whatever it tells you to) in order to automatically import those libraries.

First, we've added a class variable for the TextView. Next, in the `onCreate` method, we call the `findViewById()` method. This method takes the XML `TextView` code that we created earlier and ties it to the Java object that we created. In that method, `R.id.textView` refers to the identifier we gave to it in the XML code. The `(TextView)` type cast is needed since the method returns a generic view object.

Finally, we've added code for the `handleClick()` method. This is the method we named in the XML `android:onClick` attribute. Note that it takes a `View` parameter. This is required. That parameter gives information about the click event itself, in case you need it. Our method is very simple, though, and it only sets the text in the `TextView` to a message.

To run this program, click on the green play button or go to `Run -> Run app`. You will then have a choice of where to run your app — either on a device or using the emulator.

If you have an Android device, that is probably the quickest way to test your app. You first have to set up your Android device to be able to debug apps. As of this writing, here is how to do this: Go to the `Settings` screen and choose `About phone`. Then tap the `Build number` field 7 times. A message will pop up. Then go back and `Developer Options` will have appeared in the `Settings Menu`. Tap on it, scroll down to `Debugging` and set `USB Debugging` to be enabled. The first time you try to run a program on this phone from your computer, there will be a message about accepting a key. Make sure you accept it.

If you don't have a device, then you can use the emulator. It's a little slow to start up the emulator, and it really helps to have certain hardware settings enabled so that the emulation is reasonably fast. Try to follow the steps in the ADB

window to set up an emulator. You may have to go into your system's BIOS settings and enable a certain virtualization setting. Once you have your emulator up and running, leave it running in between tests of your code since it takes so long to start up.

Things we use a lot

- Some widgets we use a lot:

`TextView`	for displaying text
`Button`	a button
`EditText`	place to enter text

- Here is how to set text in a widget called `resultText`.

```
resultText.setText("hello")
```

- Here is how to get text from an `EditText` called `inputBox`:

```
inputBox.getText().toString();
```

- A couple of things are needed to tie your XML and Java together. First, create variables for the various widgets, like in the example below:

```
private TextView resultText;
private EditText inputBox;
```

  Then, initialize them like below. This is a little like JavaScript's `getElementById()`.

```
resultText = (TextView) findViewById(R.id.resultText);
inputBox = (EditText) findViewById(R.id.tempInput);
```

- To listen for click events on a button, in the XML, set `android:onClick="handleClick"`. Then in your Java, create the method, like below:

```
public void handleClick(View v) {
    // button handling code goes here
}
```

  I've used the name `handleClick`, but you can call it whatever. It is important that the method have a `View` parameter. That parameter has information about the click event. There are various other ways to handle click events in Android, but I've chosen to go with this for simplicity.

- Here is how to make a Toast, which is a little pop-up message at the bottom of the screen:

```
Toast.makeText(MainActivity.this, "Hello world", Toast.LENGTH_LONG).show();
```

- To help in debugging your program, you can log things to the console. To see the things you've logged, as well as any error messages, click on `Android Monitor` at the bottom of the screen and then click on `logcat`. Here is how to log things:

```
Log.v("Help!", "In handleClick() function, x= "+x);
```

  The first argument is a tag that you can use to identify your message and the second parameter is the actual message.

Intents

To start another activity from within your main activity, go to `File -> New Activity` and use the wizard to create the new activity. Then use the following (assuming the other activity is called OtherActivity) to start the other activity:

```
Intent intent = new Intent(MainActivity.this, OtherActivity.class);
startActivity(intent);
```

Special intents

To start something built into Android, like the phone dialer, camera, etc. use intents and look up the correct code. Below we open the phone dialer:

```
Intent intent = new Intent(Intent.ACTION_DIAL);
startActivity(intent);
```

Sending data to another activity

To send data to an activity you start, use something like below,

```
Intent intent = new Intent(MainActivity.this, OtherActivity.class);
intent.putExtra("keyname1", 27);
intent.putExtra("keyname2", "hello world");
startActivity(intent);
```

Then the other activity will receive this data in its `onCreate()` method, doing something like this:

```
Intent intent = getIntent();
int x = intent.getIntExtra("keyname1", 0);
String y = intent.getStringExtra("keyname2");
```

Note, the `0` in the `getIntExtra()` is a default value. There is no explicit default for `getStringExtra()`.

Receiving data from an activity you start

To receive data from an activity you start, instead of using `startActivity()`, use `startActivityForResult()`, like below:

```
Intent intent = new Intent(MainActivity.this, OtherActivity.class);
startActivityForResult(intent, 0);
```

Here is what the other activity might do to send data back:

```
Intent intent = new Intent();
intent.putExtra("keyname1", 27);
intent.putExtra("keyname2", "hello world");
setResult(RESULT_OK, intent);
finish();
```

Then the original activity uses the `onActivityResult()` method to receive the data, as shown below:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    if (resultCode != Activity.RESULT_OK)
        return;
    else {
        int x = intent.getIntExtra("keyname1", 0);
        String y = intent.getStringExtra("keyname2");
    }
}
```

The code above assumes that you call `setResult(RESULT_OK, intent)` on the intent that sends back the result. If you don't want to bother with that, then leave out the if statement entirely. You can use the `onActivityResult()` method to have particular code run when another activity returns back to the starting activity, even if the other activity is not returning any data.

Restoring the state of an activity that was restarted

If your program is restarted and you want it to remember its current state, you can do what we have below. In particular, when an Android device is rotated, it is automatically restarted, so you'll want to do something like this to

prevent it from losing all its current state.

First, override the `onSaveInstanceState()` method and put key-value pairs into the `savedInstanceState` variable in a similar way to the way we put things into intents:

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    savedInstanceState.putInt("keyname1", 27);
    savedInstanceState.putString("keyname2", "hello world");
}
```

When the app is restarted, to restore the state, notice that the `onCreate()` method has a parameter called `savedInstanceState`. Use that to get access to the key-value pairs that were stored when the app was closed. For example, in the `onCreate()` method, we could add code like this:

```
int var1;
String var2;
if (savedInstanceState != null) {
   var1 = savedInstanceState.getInt("keyname1");
   var2 = savedInstanceState.getString("keyname2");
}
else {
    var1 = 0;
    var2 = "whatever";
}
```

Threads and Timing

To have code run in the background or to run every so many milliseconds, one approach is to use threads. The code you want to run can be put into the `run()` method of a class that implements the `Runnable` interface, like below.

```
public class MyRunnable implements Runnable {
    @Override
    public void run() {
        // threaded code goes in here
    }
}
```

To then have that code run, create a `Handler` object as a class variable and use the `post()` method to start the thread.

```
handler = new Handler();
handler.post(new MyRunnable());
```

If you want the thread to run every 2 seconds, add the following line to the end of the thread's `run()` method:

```
handler.postDelayed(new MyRunnable(), 2000);
```

Including images

First, to include the image, navigate to your project's `res` directory and copy the image file into the `mipmap-mdpi` directory. There are several other directories for various screen types. In a real app, you would create images for all the different screen types, but for simplicity, we'll just copy the image into one directory and rely on Android to handle everything.

Then in the XML window, add an `ImageView` to the layout. If you do this in the Design tab, a window will pop up where you can choose the image that you uploaded. This will generate XML like the line below that links to your image (in this case a giraffe image):

```
app:srcCompat="@mipmap/giraffe"
```

If you want to resize your image, you can use something like the XML below:

```
android:maxWidth="50dp"
```

```
android:maxHeight="80dp"
android:adjustViewBounds="true"
```

ListViews

A `ListView` displays items from an array or list in a convenient vertical format. Each item in the `ListView` is clickable. The XML for a `ListView` isn't much. We just declare it and give it an ID. Then use `findViewById()` to tie the XML to a `ListView` object in the Java code. Then to tie a list or array to the `ListView`, we use something called an adapter. Here is an example that ties a `ListView` to a list/array called `names`:

```
listView.setAdapter(new ArrayAdapter<String>(this,
                                             android.R.layout.simple_list_item_1, names));
```

To handle clicks on the items of the `ListView`, we can do something like below.

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
         /* Do something with the click, relying on the pos argument to know
            which item was clicked. */
    }});
```

The code above creates an anonymous inner class to handle the click. The `pos` argument is used to know which list/array index corresponds to the click.

Databases

A database is way to store data. Android uses a particular kind called SQLite. To use it, start by creating a helper class like the one below (this is a plain Java class, not a new activity):

```
public class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, "giraffeDB", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE GIRAFFE (_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    NAME TEXT, TIME TEXT);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS GIRAFFE");
        onCreate(db);
    }

    public static void insertItem(SQLiteDatabase db, String name, String time) {
        ContentValues cv = new ContentValues();
        cv.put("NAME", name);
        cv.put("TIME", time);
        db.insert("GIRAFFE", null, cv);
    }

    public static void deleteItem(SQLiteDatabase db, int id) {
        db.delete("GIRAFFE", "_id = ?", new String[] {"" + id});
    }

    public static String getAllItemsString(SQLiteDatabase db) {
        String s = "";
        Cursor cursor = db.rawQuery("SELECT * FROM GIRAFFE", null);
        cursor.moveToFirst();
        s += cursor.getString(1) + " " + cursor.getString(2);
```

```
            while (cursor.moveToNext())
                s += "\n" + cursor.getString(1) + " " + cursor.getString(2);
            cursor.close();
            return s;
        }
    }
```

If you want to modify it to use for your own projects, you'll want to change the name `giraffeDB` to something else, and you'll want to change the table name `GIRAFFE` to something else. Our database has two columns — `NAME` and `TIME`. Be sure to replace all occurrences of those with whatever columns you decide to use.

The `onCreate()` method is called when the database is first created. The line of code that is there creates a table in the database and the columns of that table. You can also use this method to add data to the database that you want the database to start out having.

The `onUpgrade()` method is called whenever you want to move your database to a new version. The code we have here empties out the old version and calls `onCreate()` to recreate a new one.

The `insertItem()` method is called whenever someone wants to add something to the database. It uses an Android class called `ContentValues` to do its work.

The `deleteItem()` method deletes an entry from the database. The entry to be deleted is indicated by the ID that the SQLite Database automatically assigned it.

The `getAllItemsString()` method returns a string containing all the the data in the database. The code is a little tricky to follow, but basically it performs an SQL query that reads the contents of the `GIRAFFE` table and then uses a Java class called `Cursor` to read through all the results.

Then from another class, we can call the `insertItem()` method like below. Calling the other methods is done similarly.

```
SQLiteDatabase db = new DatabaseHelper(this).getReadableDatabase();
DatabaseHelper.insertItem(db, "april", "4:54 pm");
db.close();
```

Note: If your database ever gets out of sync with what you need it to be while you are developing your application, you can either delete your application from the emulator or device you are testing on or call the `onUpgrade()` method.

Tying ListViews to databases

A common task is to display items in a database inside a `ListView`. To do that use a `CursorAdapter`. Here is some code to do that:

```
SQLiteOpenHelper databaseHelper = new DatabaseHelper(context);
SQLiteDatabase db = databaseHelper.getReadableDatabase();
Cursor cursor = db.query("STUDENT", new String[]{"_id", "NAME", "GRADUATION"},
                         null, null, null, null, null);
CursorAdapter cursorAdapter = new SimpleCursorAdapter(context,
                                                      android.R.layout.simple_list_item_1,
                                                      cursor,
                                                      new String[]{"NAME"},
                                                      new int[]{android.R.id.text1},
                                                      0);
listView.setAdapter(DatabaseManager.cursorAdapter);
```

The database referenced above has columns for name and graduation year. Only the name will show up in the `ListView` as that is the argument we gave to the `CursorAdapter`. Then to listen to clicks on individual items, do something like below:

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
        Cursor c = ((SimpleCursorAdapter)listView.getAdapter()).getCursor();
        c.moveToPosition(pos);
        // do something with c.getInt(0), c.getString(1) and c.getString(2)
    }});
```

Above `c.getInt(0)` refers to the ID, `c.getString(1)` refers to the name and `c.getString(2)` refers to the graduation year (a string in this case). If the database is modified while your program is running, you'll have to refresh the cursor, like below:

```
SQLiteOpenHelper databaseHelper = new DatabaseHelper(context);
SQLiteDatabase db = databaseHelper.getReadableDatabase();
Cursor cursor = db.query("STUDENT", new String[]{"_id", "NAME", "GRADUATION"},
                         null, null, null, null, null);
cursorAdapter.swapCursor(DatabaseManager.cursor);
cursorAdapter.notifyDataSetChanged();
```

Making HTTP Requests

First, if you want to do this, you'll need to make sure you set the minimum API level of your project to 16 or higher. Second, you'll need to add a line to your Android manifest (found in the `main` subdirectory of your project) that asks for permission to use the internet on the device:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Next, add this class to your project (under File and then New Java Class). I won't try to explain how it does what it does. It's a modified version of code from *Android Programming: The Big Nerd Ranch Guide, 2nd ed.*

```
public class HTTPHelper {
    public byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();

            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                throw new IOException("error with connection!!!");
            }
            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }
    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

Then in the class that will make the HTTP request, add the following inner class:

```
private class FetchItemsTask extends AsyncTask<Void,Void,String> {
    @Override
    protected String doInBackground(Void... params) {
        String result;
        try {
            result = new HTTPHelper().getUrlString("http://www.example.com");
        } catch (IOException e) {
            result = "Error";
        }
        return result;
    }

    @Override
    protected void onPostExecute(String text) {
        textView.setText(text);
```

```
    }
}
```

This is a way of creating a background thread. Android doesn't want HTTP requests done in the main thread because such requests are slow and can cause your app to be unresponsive. So HTTP requests must happen in the background. What the code above does is call our `HTTPHelper` class to fetch the page, and the page's contents are put into a `TextView` called `textView`.

The `onPostExecute()` method takes the result returned by the `doInBackground()` method and does something with it. If we want to update any widgets on the screen with the result of the HTTP request, then we must do it in here and not elsewhere.

To create and run this background task, we would do something like this (maybe in a button handler):

```
new FetchItemsTask().execute();
```

Note that the class takes three generic parameters `<Void,Void,String>`. The first one refers to data that we might pass to the `doInBackground()` method. If we wanted to pass it some strings, we can change the `Void` to `String` and then replace the `Void ...params` with `String ...params`. This is an array of parameters, so `params[0]`, for instance, will access the first one. The second generic parameter we won't worry about here. The third one is for the return type of the `doInBackground()` method, which in this case is a String. Here is how we would modify the `FetchItemsTask` class to take in data, in this case the name of the URL to visit:

```
private class FetchItemsTask extends AsyncTask<String,Void,String> {
    @Override
    protected String doInBackground(String... params) {
        String result;
        try {
            result = new HTTPHelper().getUrlString(params[0]);
        } catch (IOException e) {
            result = "Error";
        }
        return result;
    }
     @Override
    protected void onPostExecute(String text) {
        textView.setText(text);
    }
}
```

Parsing JSON

Suppose we have the following JSON stored in a string called `s`.

```
{
"name":"Mount St. Mary's",
"details":{"enrollment":1600, "state":"Maryland"},
"majors":["math", "computer science", "cybersecurity"]
}
```

We can use the following code to read it.

```
try {
    JSONObject object = (JSONObject) new JSONObject(s);
    String name = object.getString("name");

    JSONObject details = object.getJSONObject("details");
    int enrollment = details.getInt("enrollment");

    JSONArray majors = object.getJSONArray("majors");
    String major1 = majors.getString(0);
    } catch (JSONException e) {}
```

Graphics

There are several ways to do graphics. Here is one way using canvases. First, make sure your project is set to use a minimum API level of at least 16. In the XML we just need a `LinearLayout` along with an ID to refer to it. To do the graphics, we use a combination of Android's `Canvas` and `Bitmap` objects. Declare class variables for each and then set them up like below in the `onCreate()` method:

```
linearLayout = (LinearLayout) findViewById(R.id.linearLayout);
bg = Bitmap.createBitmap(480, 800, Bitmap.Config.ARGB_8888);
canvas = new Canvas(bg);
linearLayout.setBackground(new BitmapDrawable(this.getResources(), bg));
```

Drawing will happen in a separate thread that we'll call `DrawThread`. Use a handler to start the thread:

```
handler = new Handler();
handler.post(new DrawThread());
```

Here is code for the drawing thread:

```
private class DrawThread implements Runnable {
    @Override
    public void run() {
        Paint clearPaint = new Paint();
        clearPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.CLEAR));
        canvas.drawPaint(clearPaint);

        Random random = new Random();
        int x = random.nextInt(400) + 40;
        int y = random.nextInt(720) + 40;

        Paint paint = new Paint();
        paint.setColor(Color.parseColor("#ff0000"));
        canvas.drawCircle(x, y, 40, paint);

        linearLayout.invalidate();
        handler.postDelayed(new DrawThread(), 500);
    }
}
```

This draws red circles at random locations and then erases them. The call to `invalidate()` is needed to force Android to redraw the screen. The last line schedules the next redraw to happen in 500 milliseconds. Note that there are many other things we can draw on canvases than just circles.

Touch events and gestures

Here is a really simple way to do something whenever the user touches the screen.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (event.getActionMasked() == MotionEvent.ACTION_DOWN) {
        // do something with event.getX() and event.getY()
    }
    return true;
}
```

If you need to handle more complicated gestures, have your activity implement the following interfaces: `GestureDetector.OnGestureListener` and `GestureDetector.OnDoubleTapListener`.

Then create a `GestureDetector` class variable:

```
private GestureDetector detector;
```

Initialize it in the `onCreate()` method as follows:

```
detector = new GestureDetector(this, this);
detector.setOnDoubleTapListener(this);
```

Then add all of the following. Each method is for a different kind of event. Add code to the appropriate methods to handle whatever it is you need to handle.

```
@Override
public boolean onTouchEvent(MotionEvent event){
    this.detector.onTouchEvent(event);
    return super.onTouchEvent(event);
}

@Override
public boolean onDown(MotionEvent event) {
    return true;
}

@Override
public boolean onFling(MotionEvent event1, MotionEvent event2,
                       float velocityX, float velocityY) {
    return false;
}

@Override
public void onLongPress(MotionEvent event) {
}

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
                        float distanceY) {
    return true;
}

@Override
public void onShowPress(MotionEvent event) {
}

@Override
public boolean onSingleTapUp(MotionEvent event) {
    return true;
}

@Override
public boolean onDoubleTap(MotionEvent event) {
    return true;
}

@Override
public boolean onDoubleTapEvent(MotionEvent event) {
    return true;
}

@Override
public boolean onSingleTapConfirmed(MotionEvent event) {
    return true;
}
```

Sensors

Android devices often come with a number of sensors, like an accelerometer, gyroscope, temperature sensor, etc. To use them, have your activity implement the `SensorEventListener` interface.

Then create class variables like the ones below:

```
private SensorManager sensorManager;
private Sensor sensor;
```

Initialize them as follows:

```
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

```
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
sensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_NORMAL);
```

The above uses the light sensor. Change `Sensor.TYPE_LIGHT` to whatever sensor type you want to use. Then create the following methods:

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    // do something with sensorEvent.values[]
}

@Override
public void onAccuracyChanged(Sensor arg0, int arg1) {}
```

Code to handle stuff with the sensor will go in the `onSensorChanged()` method. In particular, the `sensorEvent.values` array contains the actual data from the sensor. The `onAccuracyChanged()` method can be ignored unless you are doing sophisticated things.

Miscellaneous topics

Closing activities

To have an activity close itself, call the `finish()` method, like below:

```
finish();
```

Stuff with radio buttons

- To check if a radio button called `radio` is checked:

  ```
  if (radio.isChecked())
  ```

- To clear out any checks on a radio group called `radioGroup`:

  ```
  radioGroup.clearCheck();
  ```

Listening for keyboard events

Here is some code that listens to see if the the enter key was pressed on a pop-up keyboard associated with an `EditText` called `editText`:

```
editText.setOnEditorActionListener(new TextView.OnEditorActionListener(){
        public boolean onEditorAction(TextView exampleView, int actionId, KeyEvent event) {
            if (actionId == EditorInfo.IME_ACTION_DONE ||
                actionId == EditorInfo.IME_ACTION_UNSPECIFIED)  {
                // Do something with value in editText.getText()...
            }
            return false;
        }
   });
```

Dialog boxes

Here is some code that opens up an Okay/Cancel dialog:

```
new AlertDialog.Builder(this)
        .setTitle("Title of dialog goes here")
        .setMessage("What you want the message to be")
        .setPositiveButton(android.R.string.yes, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
               // do something if they click okay
```

```
            }
        })
         .setNegativeButton(android.R.string.no, new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int which) {
                  // do something if they click cancel
             }
         })
         .setIcon(android.R.drawable.ic_dialog_alert)
         .show();
```

Adding scrollbars

Android has a ScrollView widget that acts as a container that allows things inside to be scrollable if there is too much stuff to fit on the screen. Here is what a snippet of the XML might look like:

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

         <TextView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:text="Hello world"/>
    </LinearLayout>
</ScrollView>
```

Using the Camera flash

First, you can add the following to your Android manifest file to request permission to use the camera and flash.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />

<uses-feature android:name="android.hardware.camera" android:required="false" />
<uses-feature android:name="android.hardware.camera.flash" android:required="false" />
```

Next, the lines below open the camera for use:

```
Camera cam = Camera.open();
Camera.Parameters p = cam.getParameters();
```

To set the flash on, use the following:

```
p.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
cam.setParameters(p);
cam.startPreview();
```

To turn the flash off, use the above, but use `FLASH_MODE_OFF` instead of `FLASH_MODE_TORCH`. When you are done with the camera, use the following to close it:

```
cam.stopPreview();
cam.release();
```

Getting the text from a clicked button

Suppose we have an event handler that handles clicks from several buttons and we want to get the button's text. We can use the `View` parameter of the event handler to do this, but we first have to cast the abstract `View` to something concrete, like a Button object. See below:

```
public void handleClick(View v) {
    Button b = (Button) v;
    // now do something with b.getText()
}
```

TableLayout

A `TableLayout` allows things to be laid out in tabular form. Use `<TableRow>` to start a new row. Here is an example that creates a table with one row of two buttons. Add more instances of `<TableRow>` inside the layout to add more rows.

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
</TableLayout>
```

Reading from a text file

First, in the `res` subfolder of your project, create a folder called `raw` and place the text file in there. You may need to clean the project (the option is under the `Build` menu in Android Studio) in order for the file to be recognized. The following function will read the contents of the file into a string. Replace the reference to `R.raw.textFile` with something suitable to your program.

```
public String readFile() {
    String s = "";
    try {
        String line;
        BufferedReader reader = new BufferedReader(new
         InputStreamReader(this.getResources().openRawResource(R.raw.textFile)));
        while ((line = reader.readLine()) != null)
            s += line + "\n";
        reader.close();
    } catch (IOException e) { e.printStackTrace(); }
    return s;
}
```