MobIle devIceS
CME 4423

Dr. M. Hilal Özcanhan

Fall 2012

Mobile Devices More Than An Internet Tool

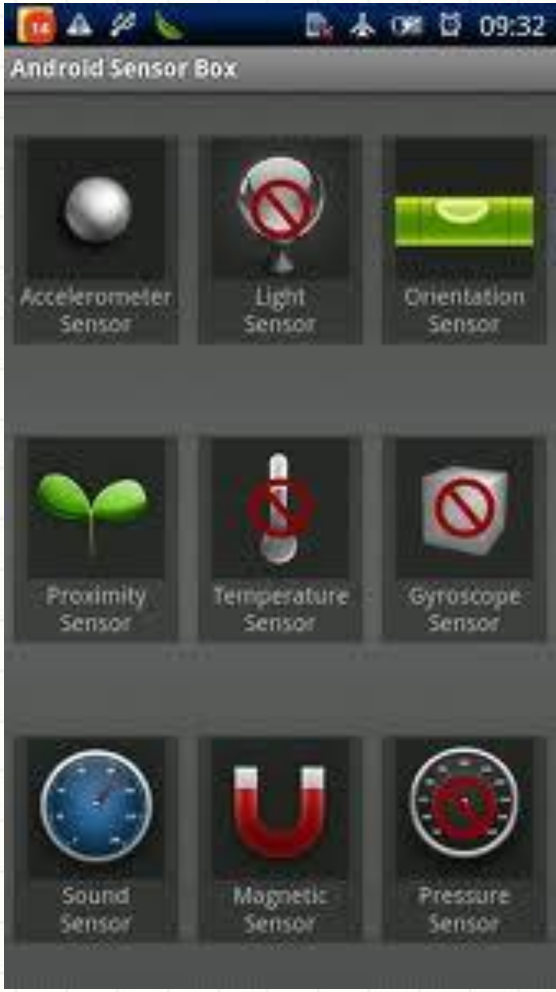
Modern mobile phones → much more than a simple device with a connection to the Internet.

  - Microphones, cameras, accelerometers, compasses, temperature gauges, and brightness detectors

- Smart-phones have become extra-sensory devices, able to ***augment perceptions***.

Sensors

- Sensors detect physical and environmental properties
- Enhance the user experience of mobile applications.
  - Electronic compasses, gravity sensors, brightness gauges, proximity sensors, temperature, gyroscope, pressure, orientation, accelerometer



Sensors

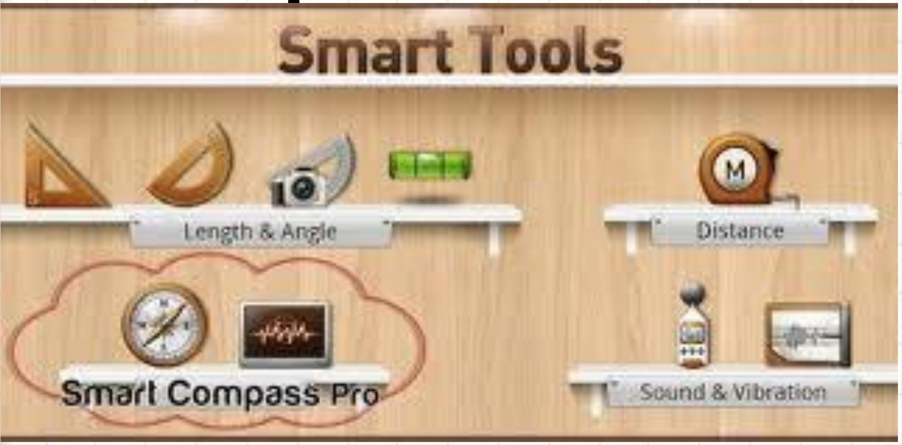
- Interacting with devices with reality and physical movement-based input.

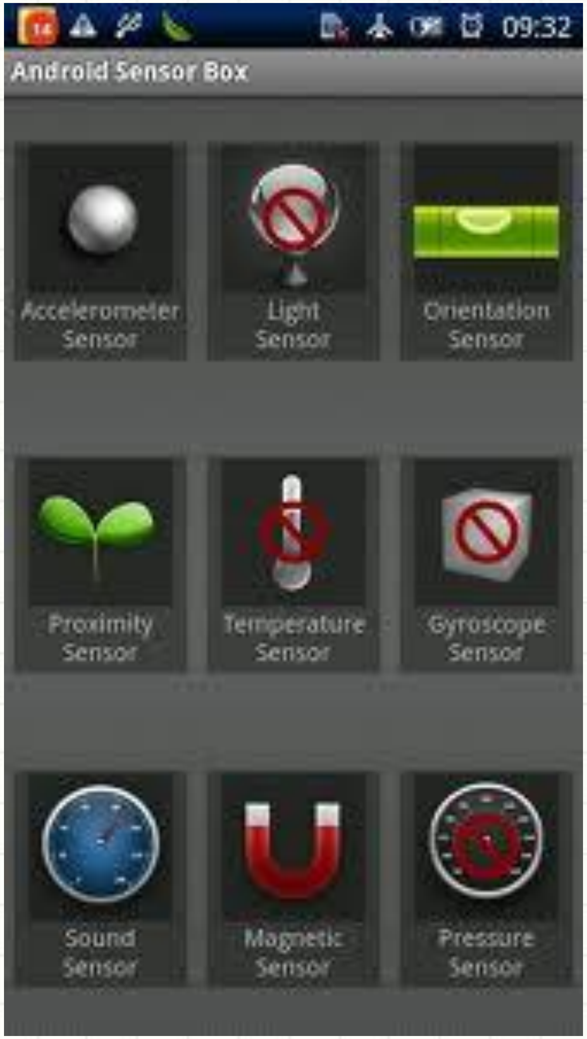
Acceleration

Distance

Vibrations

Compass





Available Sensors

The **Sensor Manager** is used to manage the sensors available on Android devices.

getSystemService returns a reference to the Sensor Manager Service:

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

The Sensor Class

The Sensor class describes the properties of each hardware sensor: type, name, manufacturer, accuracy and range.

The Sensor class includes constants that describes the type of hardware sensor represented by a Sensor object:

Sensor.TYPE_<TYPE>.

Supported Android Sensors

- Sensor.TYPE_ACCELEROMETER: three-axis accelerometer sensor, returns the current acceleration along three axes in m/s^2.
- Sensor.TYPE_GYROSCOPE: gyroscopic sensor, returns the current device orientation on three axes in degrees.
- Sensor.TYPE_LIGHT: light sensor that returns a single value describing the ambient illumination in lux. used to dynamically control the screen brightness.
- Sensor.TYPE_MAGNETIC_FIELD: magnetic field sensor that finds the current magnetic field in microteslas along three axes.
- Sensor.TYPE_ORIENTATION: orientation sensor that returns the device orientation on three axes in degrees.
- Sensor.TYPE_PRESSURE: pressure sensor that returns a single value, the current pressure exerted on the device in kilopascals.
- Sensor.TYPE_PROXIMITY: proximity sensor that indicates the distance between the device and the target object in meters.
- Sensor.TYPE_TEMPERATURE: thermometer that returns temperature in degrees Celsius. The temperature returned may be the room temperature, device battery temperature, or remote sensor temperature.

Default Sensor

- Use the Sensor Manager’s ***getDefaultSensor***, passing in the sensor-type :

```
Sensor defaultGyroscope =
sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

- If no default Sensor exists for the given type, the method returns null.

List of Available Sensors

- Use ***getSensorList*** to return the available pressure sensor objects:

```
List<Sensor> pressureSensors =
sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
```

- To find every Sensor available getSensorList, passing in Sensor.TYPE_ALL,

```
List<Sensor> allSensors =
sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Using Sensors: Sensor Event Listener

```
final SensorEventListener mySensorEventListener = new SensorEventListener() {
                        public void onSensorChanged(SensorEvent sensorEvent) {
                                    // TO Monitor Sensor changes.
                                                                             }
            public void onAccuracyChanged(Sensor sensor, int accuracy) {
                                    // TO React to a change in Sensor accuracy.
                                                                                        }
                                                                                                  };
```

SensorEvent parameter in the onSensorChanged method includes four properties To describe a Sensor event:

- sensor : The Sensor object that triggered the event.
- accuracy : The accuracy of the Sensor when the event occurred (low, medium, high, or unreliable).
- values : A float array that contains the new value(s) detected.
- timestamp : The time (in nanoseconds) at which the Sensor event occurred.

Accuracy of a Sensor

onAccuracyChanged method gives the accuracy value from the monitored Sensor’s accuracy :

➤ SensorManager.SENSOR_STATUS_ACCURACY_LOW

Sensor is reporting low accuracy and needs to be calibrated

➤ SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM

Sensor data is average accuracy, and calibration might improve readings

➤ SensorManager.SENSOR_STATUS_ACCURACY_HIGH

Sensor is reporting highest possible accuracy

➤ SensorManager.SENSOR_STATUS_UNRELIABLE

Sensor data is unreliable, either calibration is required or readings are not currently possible

Receiving Sensor events

- Register: Sensor Event Listener with the Sensor Manager.
- Specify Sensor object: to observe, and the rate of receiving updates.
- Example of registering Sensor Event Listener for the default proximity Sensor at the normal update rate:

```
Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
sensorManager.registerListener(mySensorEventListener, sensor,
                                        SensorManager.SENSOR_DELAY_NORMAL);
```

Sensor Manager constants

- In descending order of responsiveness:
  - ➢ SensorManager.SENSOR_DELAY_FASTEST

Specifies the fastest possible Sensor update rate

- ➢ SensorManager.SENSOR_DELAY_GAME

Selects an update rate suitable for use in controlling games

- ➢ SensorManager.SENSOR_DELAY_NORMAL

Specifies the default update rate

- ➢ SensorManager.SENSOR_DELAY_UI

Specifies a rate suitable for updating UI features

- The rate selected is not binding; the Sensor Manager may return results faster or slower than specified, though it will tend to be faster.
- To minimize the associated resource cost of using the Sensor in the application try to select the slowest suitable rate.

Unregistering Sensor Event Listeners

It’s important to unregister the Sensor Event Listeners when the application no longer needs to receive updates

SensorManager.unregisterListener(mySensorEventListener);

It’s good practice to register and unregister Sensor Event Listener in the *onResume* and *onPause*

methods of the **Activities** to ensure they’re being used only when the Activity is active.

INTERPRETING SENSOR VALUES

The length and composition of values returned in the onSensorChanged event depend on the Sensor being monitored.

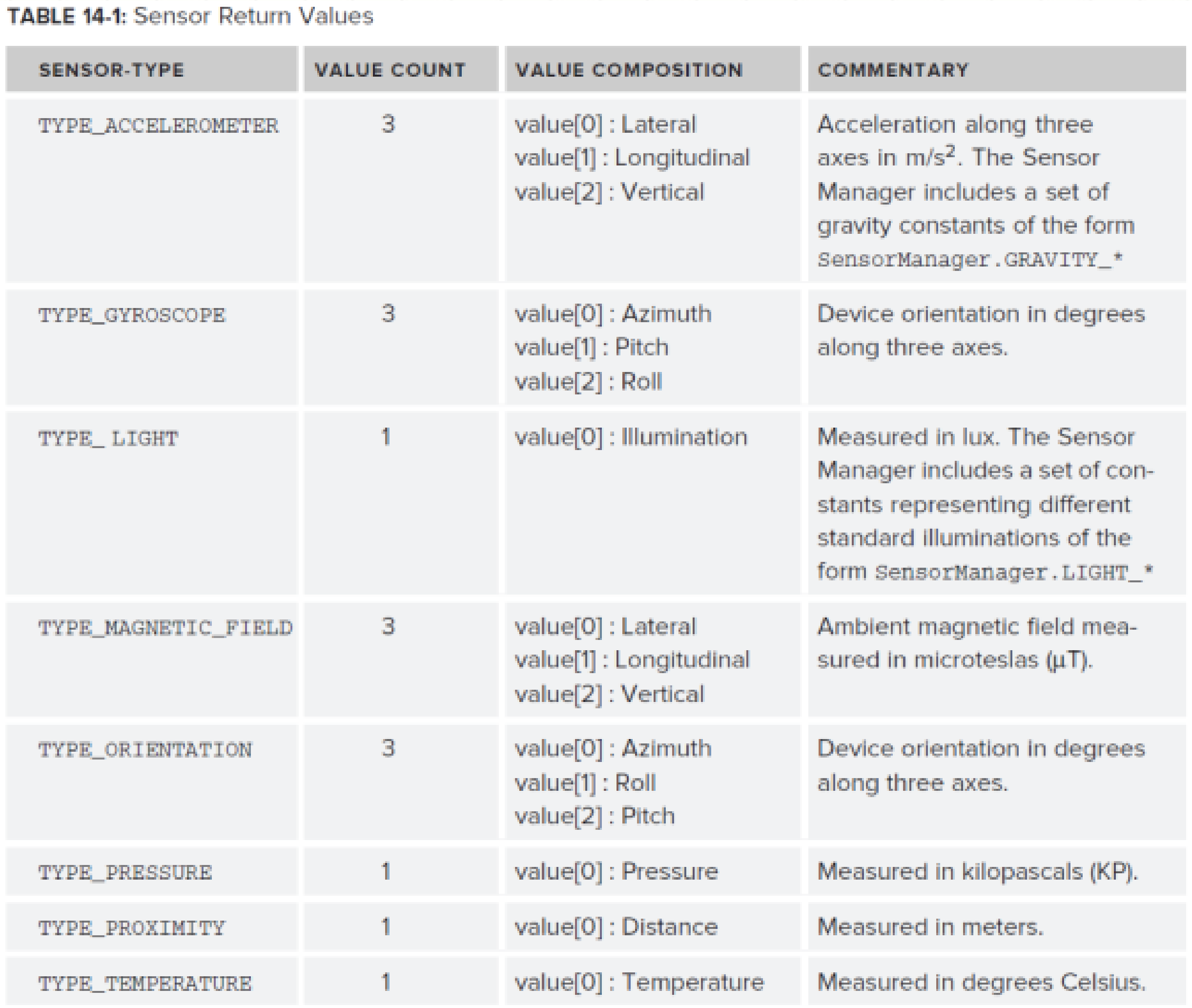


TABLE 14-1: Sensor Return Values

SENSOR-TYPE	VALUE COUNT	VALUE COMPOSITION	COMMENTARY
TYPE_ACCELEROMETER	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Acceleration along three axes in m/s^2. The Sensor Manager includes a set of gravity constants of the form SensorManager.GRAVITY_*
TYPE_GYROSCOPE	3	value[0] : Azimuth value[1] : Pitch value[2] : Roll	Device orientation in degrees along three axes.
TYPE_ LIGHT	1	value[0] : Illumination	Measured in lux. The Sensor Manager includes a set of constants representing different standard illuminations of the form SensorManager.LIGHT_*
TYPE_MAGNETIC_FIELD	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Ambient magnetic field measured in microteslas (μT).
TYPE_ORIENTATION	3	value[0] : Azimuth value[1] : Roll value[2] : Pitch	Device orientation in degrees along three axes.
TYPE_PRESSURE	1	value[0] : Pressure	Measured in kilopascals (KP).
TYPE_PROXIMITY	1	value[0] : Distance	Measured in meters.
TYPE_TEMPERATURE	1	value[0] : Temperature	Measured in degrees Celsius.

USING COMPASS, ACCELEROMETER & ORIENTATION SENSORS

- For Movement and orientation in applications: Orientation and accelerometer sensors.
- Accelerometers and compasses are used to provide functionality on device direction, orientation, and movement.
- A recent trend : Use this functionality to provide input mechanisms other than touchscreen, trackball, and keyboard.

Availability of Sensors

- The availability of compass and accelerometer Sensors depends on the hardware on which the application runs.
- Always check for the availability of any required Sensors and make sure applications fail gracefully if they are missing.
- When available, they are reached through the Sensor Manager.

Sensor Capabilities

- When available the compass and accelerometer sensors allow the following:

➢ Determine the current device orientation

➢ Monitor and track changes in orientation

➢ Know which direction the user is facing

➢ Monitor acceleration—changes in movement rate— vertically, laterally, or longitudinally

Uses of orientation, direction, movement:

Monitoring orientation, direction, and movement:

➢ Use the compass and accelerometer to determine speed and direction. With a map, camera, and location-based services to augment location based data over real-time camera feed.

➢ Create user interfaces that adjust dynamically to suit the orientation of the device. Android alters screen orientation.

➢ Monitor for rapid acceleration to detect drop or throw.

➢ Measure movement or vibration.

lock the device; any movement - send an alert SMS

➢ Create user interface controls that use physical gestures and movement as input.

Accelerometers

- Acceleration, the rate of change of velocity
- Also referred as gravity sensors
- *At rest :* SensorManager.STANDARD_GRAVITY *constant, 9.8m/s2*
- *Does* not *measure velocity, can't measure speed directly based on a single accelerometer reading.*
- *Need to measure changes in acceleration <u>over time</u>*.

Accelerometers

- Need to calibrate the device to note the initial orientation and acceleration.
- Sensor Manager reports accelerometer Sensor changes along all three axes.
- The values passed in through the values property of the Sensor Event Listener’s Sensor Event lateral, longitudinal, and vertical.

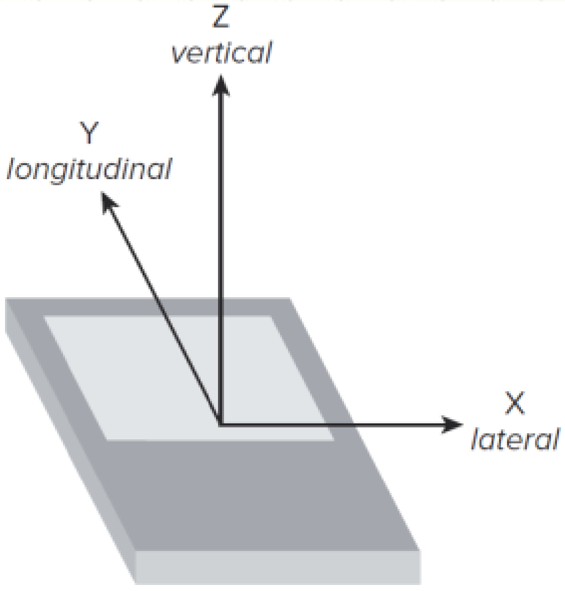
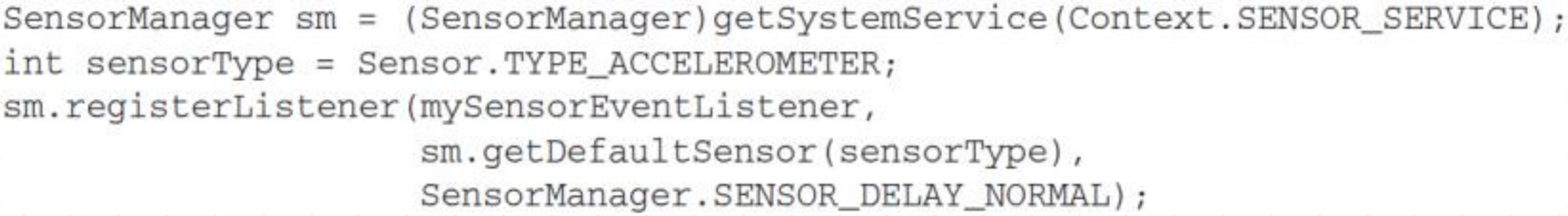


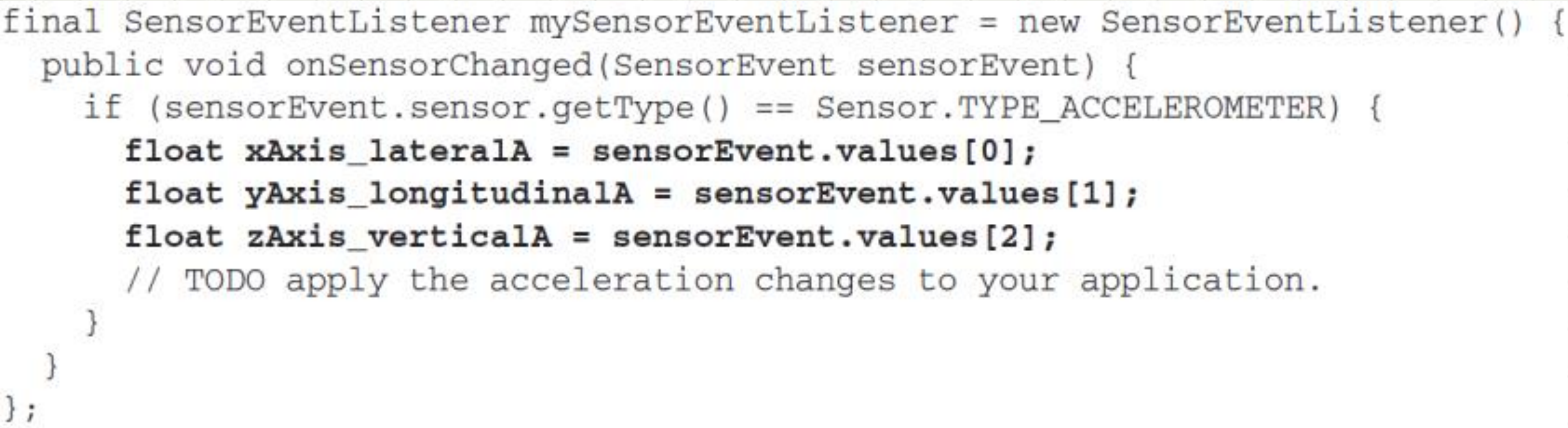
FIGURE 14-1

Listening to changes to default accelerometer

- Sensor Listener should implement the onSensorChanged method fired when acceleration in any direction is measured.



```
SensorManager sm = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
int sensorType = Sensor.TYPE_ACCELEROMETER;
sm.registerListener(mySensorEventListener,
                    sm.getDefaultSensor(sensorType),
                    SensorManager.SENSOR_DELAY_NORMAL);
```

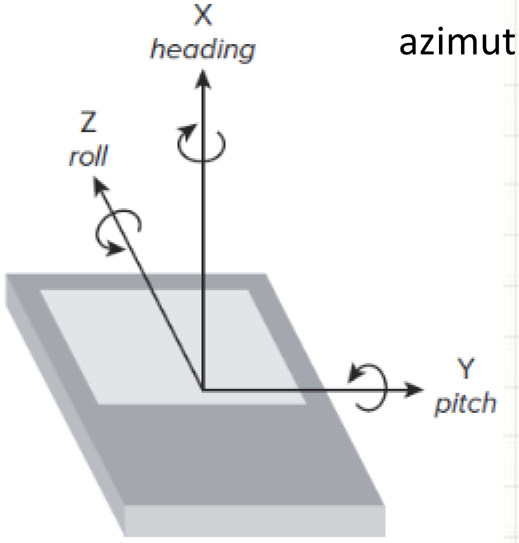


```
final SensorEventListener mySensorEventListener = new SensorEventListener() {
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
      float xAxis_lateralA = sensorEvent.values[0];
      float yAxis_longitudinalA = sensorEvent.values[1];
      float zAxis_verticalA = sensorEvent.values[2];
      // TODO apply the acceleration changes to your application.
    }
  }
};
```

- The onSensorChanged method receives a SensorEvent that includes a float array containing the acceleration measured along all three axes: lateral, longitudinal, vertical

Determining Orientation

- Device orientation is calculated based on the accelerometer and magnetic field along all three axes.
- Orientation Sensor is a combination of the magnetic field Sensors (electronic compass) and accelerometers: the pitch and roll.



azimuth , heading or yaw

FIGURE 14-2

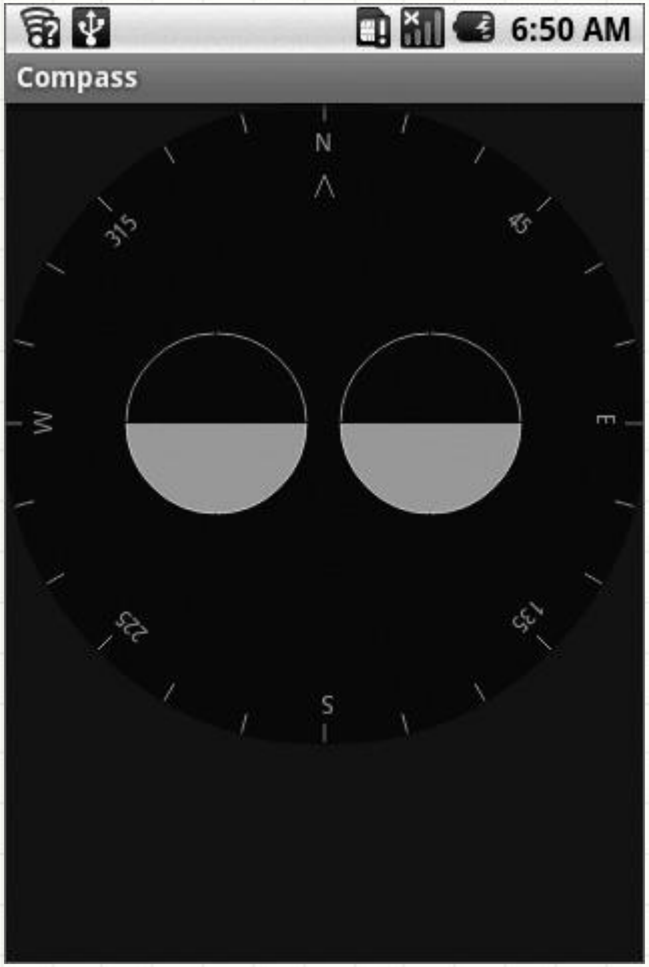
Determining Orientation Using Two Sensors

Calculate Orientation Using Accelerometer and Magnetic Field Sensors

- Use both the accelerometer and magnetic field Sensors:
- Create and register two Sensor Event Listeners.
- Make calculations and **Remap the Orientation Reference Frame**

Creating a Compass and Artificial Horizon

- 10 steps:
- CompassView to experiment with owner-drawn controls.
- Extend the functionality of the Compass View to display the device pitch and roll.



CONTROLLING DEVICE VIBRATION

- Creating Notifications in Ch. 9
- Use vibration to enrich event feedback.
- Vibrating the device is an excellent way to provide user feedback, and is particularly popular as a feedback mechanism for games.

Adding Vibration Permission

- To control device vibration, applications need the VIBRATE permission.
- Add to Notification application manifest the following XML snippet:

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

Device vibration

- Device vibration is controlled through the Vibrator Service, accessible via the getSystemService method:

```
String vibratorService = Context.VIBRATOR_SERVICE;
Vibrator vibrator = (Vibrator)getSystemService(vibratorService);
```

- Call vibrate to start device vibration
- Pass in either a vibration duration or a pattern of alternating vibration/pause sequences :

```
long[] pattern = {1000, 2000, 4000, 8000, 16000 };
vibrator.vibrate(pattern, 0); // Execute vibration pattern.
vibrator.vibrate(1000); // Vibrate for 1 second.
```

- To cancel vibration call cancel;