

# CMSC436: Programming Handheld Systems

# Sensors

# Today's Topics

Sensor & SensorManager

SensorEvent & SensorEventListener

Filtering sensor values

Example applications

# Sensors

Hardware devices that measure the physical environment

Motion

Position

Environment

# Some Example Sensors

Motion - 3-axis Accelerometer

Position - 3-axis Magnetic field

Environment - Pressure

# Sensor Types

int TYPE\_MOTION\_DETECT

int TYPE\_GRAVITY

int TYPE\_AMBIENT\_TEMPERATURE

int TYPE\_ACCELEROMETER

int TYPE\_ALL

# Some Sensor Methods

float getResolution()

float getPower()

int getReportingMode()

int getMinDelay()

float getMaximumRange()

# SensorEvent

Represents a Sensor event

Data includes

- Sensor type

- Time-stamp

- Accuracy

- Sensor-specific measurement data



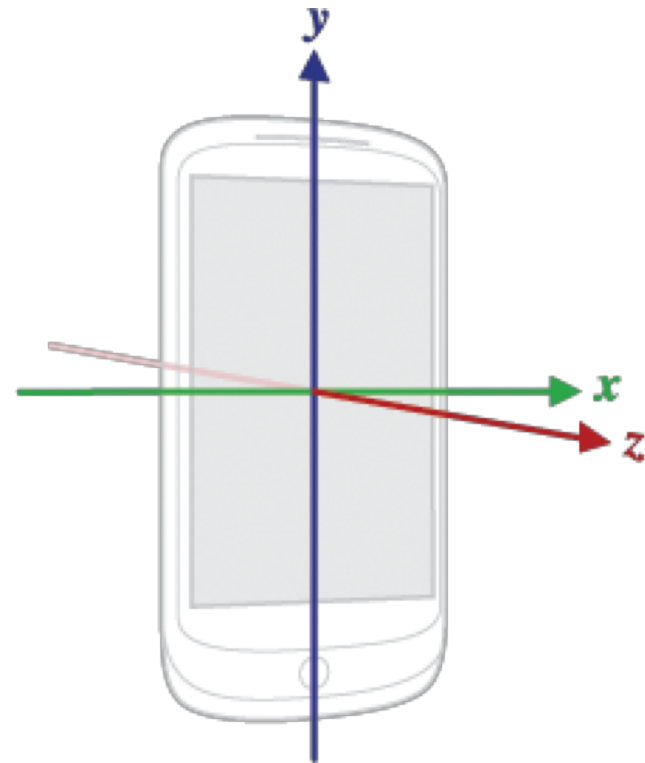
# Sensor Coordinate System

When default orientation is portrait & the device is lying flat, face-up on a table, axes run

X – Left to right

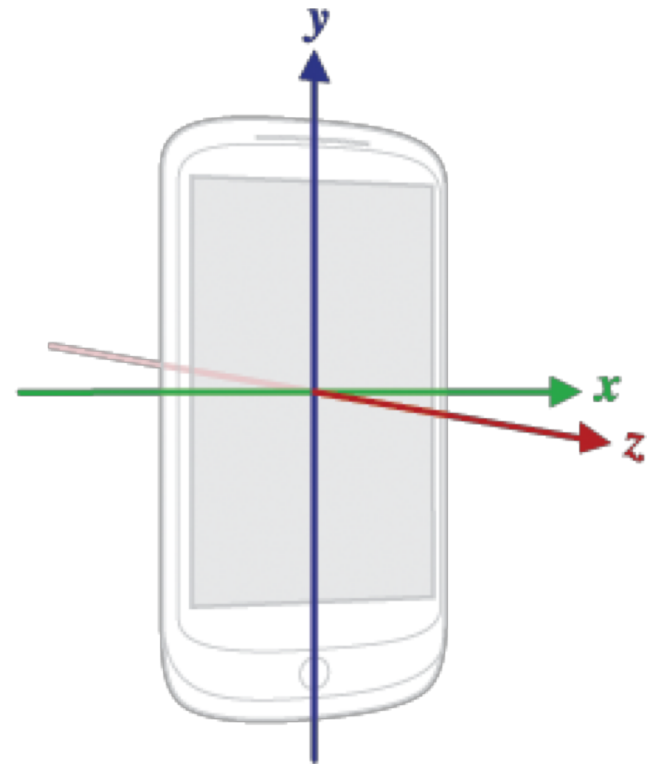
Y – Bottom to top

Z – Down to up



# Sensor Coordinate System

Coordinate system  
does not change when  
device orientation  
changes



# SensorManager

System service that manages sensors

Get instance with

```
getSystemService(Context.SENSOR_SERVICE )
```

Access a specific sensor with

```
SensorManager.getDefaultSensor(int type)
```

# Some Sensor Type Constants

Accelerometer - `Sensor.TYPE_ACCELEROMETER`

Magnetic field - `Sensor.TYPE_MAGNETIC_FIELD`

Pressure – `Sensor.TYPE_PRESSURE`

# Some SensorManager Methods

List<Sensor> getSensorList (int type)

Sensor getDefaultSensor (int type)

# SensorEventListener

Interface for SensorEvent callbacks

# SensorEventListener

Called when a sensor's accuracy has changed

```
void onAccuracyChanged(  
    Sensor sensor, int accuracy)
```

# Accuracy Constants

SENSOR\_STATUS\_ACCURACY\_HIGH

SENSOR\_STATUS\_ACCURACY\_MEDIUM

SENSOR\_STATUS\_ACCURACY\_LOW

SENSOR\_STATUS\_NO\_CONTACT

SENSOR\_STATUS\_UNRELIABLE



# SensorEventListener

Called when sensor values have changed

```
void onSensorChanged(SensorEvent event)
```

Note: This method should not keep a reference to the SensorEvent

# Registering for SensorEvents

Use the SensorManager to register/unregister for SensorEvents

# Registering for SensorEvents

Register SensorEventListener for a given sensor

```
public boolean registerListener (  
    SensorEventListener listener,  
    Sensor sensor, int rate)
```

# Registering for SensorEvents

Unregisters a listener for the sensors with which it is registered

```
public void unregisterListener (  
    SensorEventListener listener,  
    Sensor sensor)
```

# SensorRawAccelerometer

Displays the raw values read from the device's accelerometer

Extended controls

Accelerometer Additional sensors

Location

Cellular

Battery

Phone

Directional pad

Microphone

Fingerprint

Virtual sensors

Bug report

Settings

Help

Rotate Move

Yaw -180 180 0.0

Pitch -180 180 0.0

Roll -180 180 0.0

Device rotation

Resulting values

Accelerometer (m/s<sup>2</sup>): 0.00 9.81 0.00

Gyroscope (rad/s): 0.00 0.00 0.00

Magnetometer (µT): 22.00 5.90 43.10

Rotation: ROTATION\_0

SensorRawAccelerometer

Raw X: 0.0


Raw Y: 9.81

Raw Z: 0.0

SensorRawAccelerometer

Extended controls

Accelerometer Additional sensors




Rotate  Move

Yaw  34.4

Pitch  -26.5

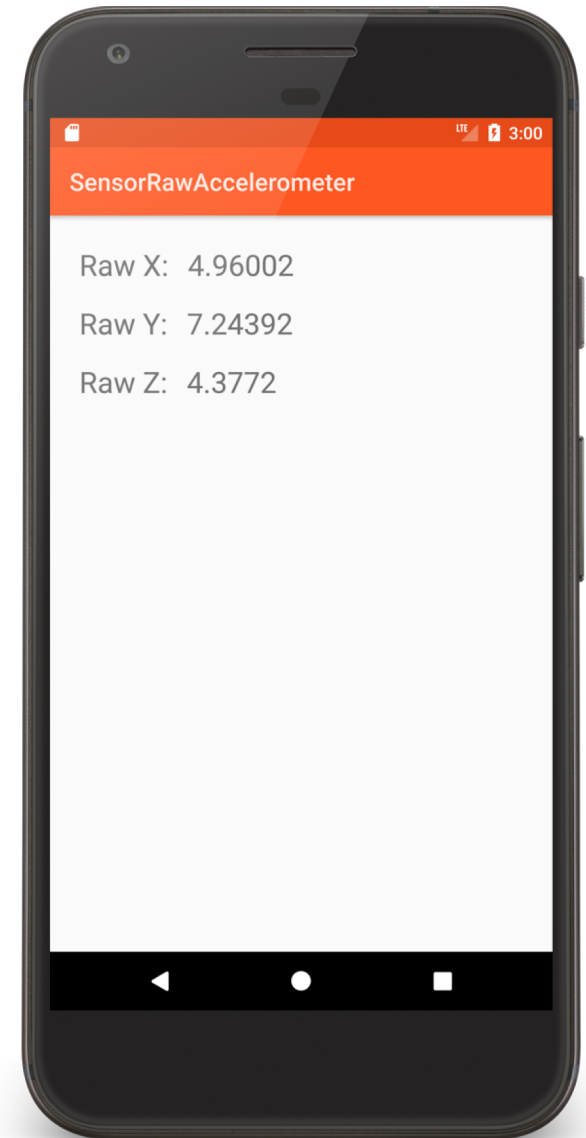
Roll  31.8

Device rotation



Resulting values

Accelerometer (m/s <sup>2</sup> ):	4.96	7.24	4.38
Gyroscope (rad/s):	0.00	0.00	-0.00
Magnetometer (μT):	-12.49	-11.13	45.79
Rotation:	ROTATION_0		



```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    mXValueView = findViewById(R.id.x_value_view);  
    mYValueView = findViewById(R.id.y_value_view);  
    mZValueView = findViewById(R.id.z_value_view);  
  
    // Get reference to SensorManager  
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);  
  
    // Get reference to Accelerometer  
    if (null != mSensorManager) {  
        mAccelerometer = mSensorManager  
            .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);  
    }  
    if (null == mAccelerometer) finish();  
}
```



*// Register listener*

```
protected void onResume() {  
    super.onResume();  
    mSensorManager.registerListener(this, mAccelerometer,  
        SensorManager.SENSOR_DELAY_UI);  
    mLastUpdate = System.currentTimeMillis();  
}
```

*// Unregister listener*

```
protected void onPause() {  
    mSensorManager.unregisterListener(this);  
    super.onPause();  
}
```

*// Process new reading*

```
public void onSensorChanged(SensorEvent event) {  
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {  
        long actualTime = System.currentTimeMillis();  
        if (actualTime - mLastUpdate > UPDATE_THRESHOLD) {  
            mLastUpdate = actualTime;  
            float x = event.values[0], y = event.values[1], z = event.values[2];  
            // update values on display  
        }  
    }  
}
```

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {  
    // Not implemented  
}
```

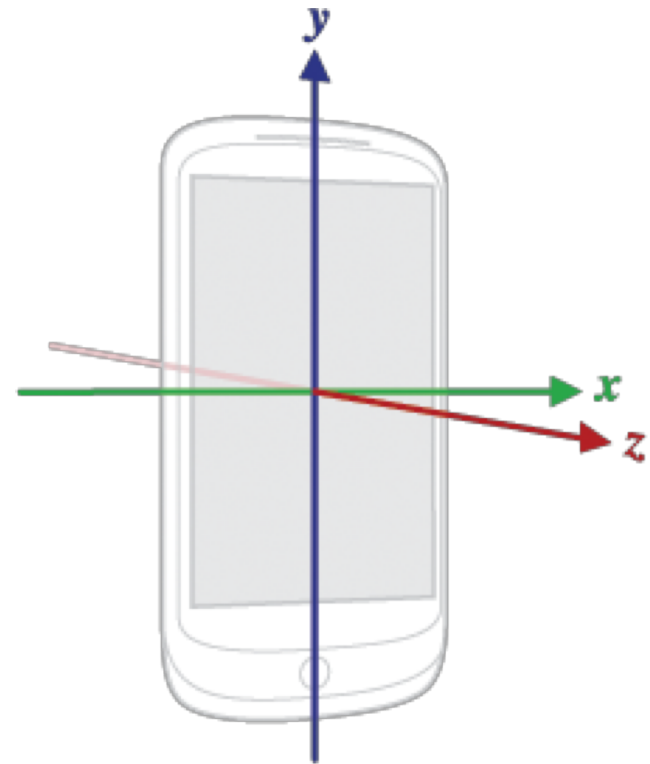
# Accelerometer Values

If the device were standing straight up, the accelerometer would ideally report:

$$X \approx 0 \text{ m/s}^2$$

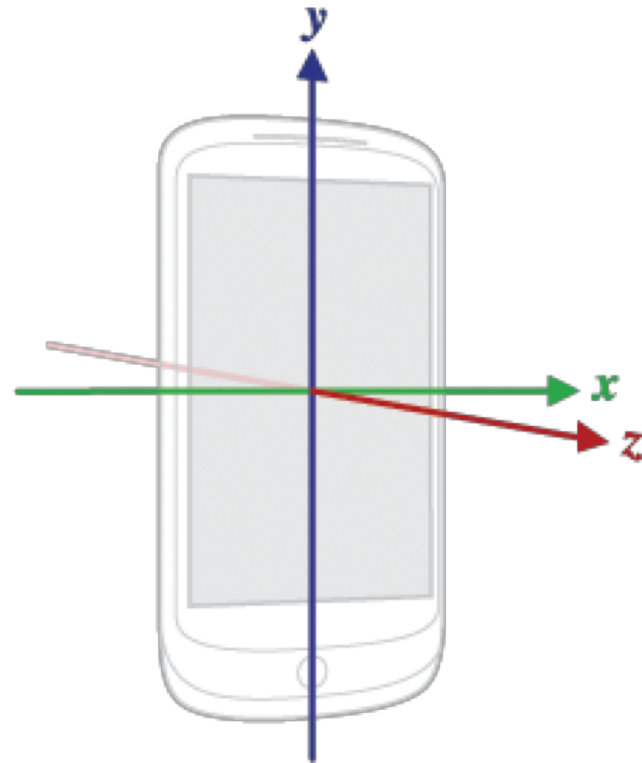
$$Y \approx 9.81 \text{ m/s}^2$$

$$Z \approx 0 \text{ m/s}^2$$



# Accelerometer values

But these values will vary due to natural movements, non-flat surfaces, noise, etc.



# Filtering Accelerometer Values

Two common transforms

- Low-pass filter

- High-pass filter

# Low-Pass Filter

Deemphasize transient force changes

Emphasize constant force components



Carpenter's Level

# High-Pass Filter

Emphasize transient force changes

Deemphasize constant force components





Percussion  
Instrument

# SensorFilteredAccelerometer


Applies both a low-pass and a high-pass filter to raw accelerometer values

Displays the filtered values

Extended controls

Location  
Cellular  
Battery  
Phone  
Directional pad  
Microphone  
Fingerprint  
Virtual sensors  
Bug report  
Settings  
Help

Accelerometer Additional sensors




Rotate  Move

X  0.0

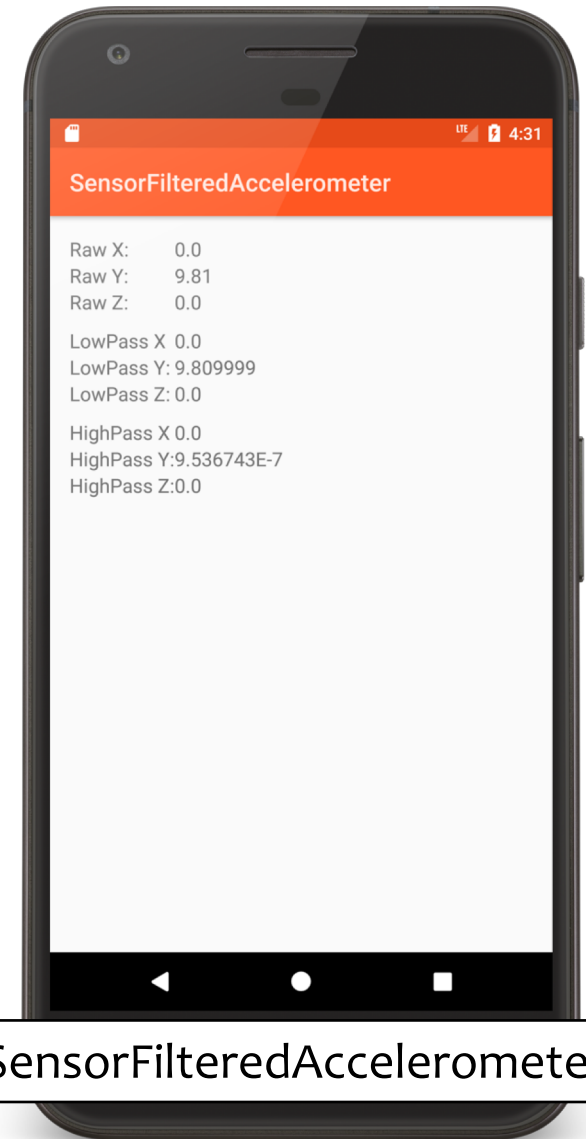
Y  0.0

Device rotation



Resulting values

Accelerometer (m/s <sup>2</sup> ):	0.00	9.81	0.00
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	22.00	5.90	43.10
Rotation:	ROTATION_0		




SensorFilteredAccelerometer

Extended controls

- Location
- Cellular
- Battery
- Phone
- Directional pad
- Microphone
- Fingerprint
- Virtual sensors
- Bug report
- Settings
- Help

Accelerometer Additional sensors




Rotate
  Move

X  -1.9

Y  -0.7

Device rotation



Resulting values

Accelerometer (m/s <sup>2</sup> ):	0.00	9.81	0.00
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	22.00	5.90	43.10
Rotation:	ROTATION_0		

4:33

### SensorFilteredAccelerometer

Raw X: 0.0  
Raw Y: 9.81  
Raw Z: 0.0

LowPass X 0.13405488  
LowPass Y: 9.913117  
LowPass Z: 0.0

HighPass X -0.13405488  
HighPass Y:-0.10311699  
HighPass Z:0.0

```
public void onCreate(Bundle savedInstanceState) {  
    ...  
    // Get reference to SensorManager  
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);  
  
    if (null != mSensorManager) {  
        // Get reference to Accelerometer  
        mAccelerometer = mSensorManager  
            .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);  
        if (null == mAccelerometer) finish();  
  
        mLastUpdate = System.currentTimeMillis();  
    }  
}
```

*// Register listener*

```
protected void onResume() {  
    super.onResume();  
    mSensorManager.registerListener(this, mAccelerometer,  
        SensorManager.SENSOR_DELAY_UI);  
}
```

*// Unregister listener*

```
protected void onPause() {  
    super.onPause();  
    mSensorManager.unregisterListener(this);  
}
```

*// Process new reading*

```
public void onSensorChanged(SensorEvent event) {  
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {  
        long actualTime = System.currentTimeMillis();  
        if (actualTime - mLastUpdate > 500) {  
            mLastUpdate = actualTime;  
  
            float rawX = event.values[0];  
            float rawY = event.values[1];  
            float rawZ = event.values[2];
```

*// Apply low-pass filter*

```
mGravity[0] = lowPass(rawX, mGravity[0]);  
mGravity[1] = lowPass(rawY, mGravity[1]);  
mGravity[2] = lowPass(rawZ, mGravity[2]);
```

*// Apply high-pass filter*

**mAccel[0]** = highPass(rawX, **mGravity[0]**);

**mAccel[1]** = highPass(rawY, **mGravity[1]**);

**mAccel[2]** = highPass(rawZ, **mGravity[2]**);

**mXValueView**.setText(String.valueOf(rawX));

**mYValueView**.setText(String.valueOf(rawY));

**mZValueView**.setText(String.valueOf(rawZ));

**mXGravityView**.setText(String.valueOf(**mGravity[0]**));

**mYGravityView**.setText(String.valueOf(**mGravity[1]**));

**mZGravityView**.setText(String.valueOf(**mGravity[2]**));

**mXAccelView**.setText(String.valueOf(**mAccel[0]**));

**mYAccelView**.setText(String.valueOf(**mAccel[1]**));

**mZAccelView**.setText(String.valueOf(**mAccel[2]**));



*// Deemphasize transient forces*

```
private float lowPass(float current, float gravity) {  
    float mAlpha = 0.8f;  
    return gravity * mAlpha + current * (1 - mAlpha);  
}
```

*// Deemphasize constant forces*

```
private float highPass(float current, float gravity) {  
    return current - gravity;  
}
```


# SensorCompass

Uses the device's accelerometer and magnetometer to orient a compass

Extended controls

- Location
- Cellular
- Battery
- Phone
- Directional pad
- Microphone
- Fingerprint
- Virtual sensors
- Bug report
- Settings
- Help

Accelerometer Additional sensors



Rotate
  Move

Yaw  -180 180 **-180.0**

Pitch  -180 180 **-45.0**

Roll  -180 180 **0.0**

Device rotation

Resulting values

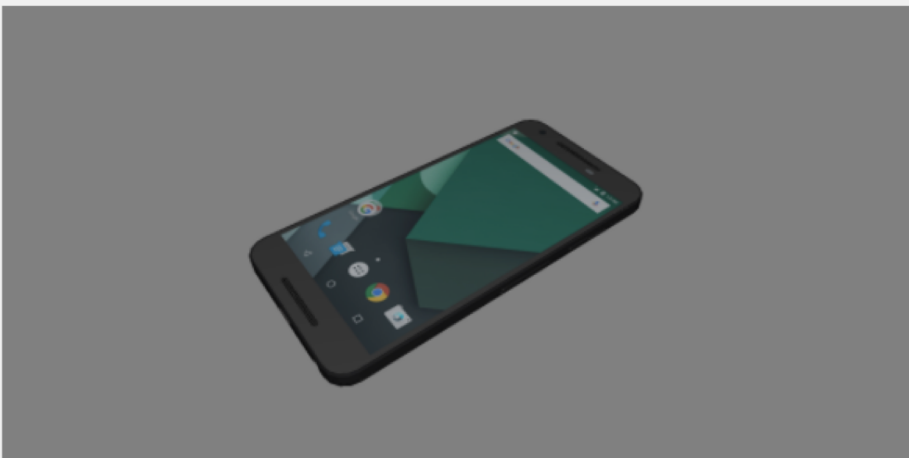
Accelerometer (m/s <sup>2</sup> ):	0.00	-6.94	6.94
Gyroscope (rad/s):	-0.00	0.00	-0.00
Magnetometer (μT):	-22.00	26.30	34.65
Rotation:	ROTATION_0		



Extended controls

- Location
- Cellular
- Battery
- Phone
- Directional pad
- Microphone
- Fingerprint
- Virtual sensors
- Bug report
- Settings
- Help

Accelerometer Additional sensors



Rotate
  Move

Yaw  -52.9  
 Pitch  -45.0  
 Roll  0.0

Device rotation

Resulting values

Accelerometer (m/s <sup>2</sup> ):	-5.53	4.18	6.94
Gyroscope (rad/s):	0.00	-0.00	-0.00
Magnetometer (μT):	34.25	1.68	34.65
Rotation:	ROTATION_0		



```
protected void onCreate(Bundle savedInstanceState) {
```

```
    ...
```

```
    // Get a reference to the SensorManager
```

```
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

```
    if (null != mSensorManager) {
```

```
        // Get a reference to the accelerometer
```

```
        accelerometer =
```

```
            mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

```
        // Get a reference to the magnetometer
```

```
        magnetometer =
```

```
            mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

```
    }
```

```
    // Exit unless both sensors are available
```

```
    if (null == accelerometer || null == magnetometer)
```

```
        finish();
```

```
}
```

```
onResume() {  
    super.onResume();  
  
    // Register for sensor updates  
    mSensorManager.registerListener(this, accelerometer,  
        SensorManager.SENSOR_DELAY_NORMAL);  
    mSensorManager.registerListener(this, magnetometer,  
        SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
protected void onPause() {  
    super.onPause();  
  
    // Unregister all sensors  
    mSensorManager.unregisterListener(this);  
}
```

```
public void onSensorChanged(SensorEvent event) {  
    // Acquire accelerometer event data  
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {  
        mGravity = new float[3];  
        System.arraycopy(event.values, 0, mGravity, 0, 3);  
    }  
    // Acquire magnetometer event data  
    else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {  
        mGeomagnetic = new float[3];  
        System.arraycopy(event.values, 0, mGeomagnetic, 0, 3);  
    }  
  
    // If we have readings from both sensors then use the readings to compute the  
    // device's orientation and then update the display.  
    if (mGravity != null && mGeomagnetic != null) {  
        float rotationMatrix[] = new float[9];  
    }  
    ...  
}
```

```
// Users the accelerometer and magnetometer readings to compute the device's  
// rotation with respect to a real-world coordinate system  
boolean success = SensorManager.getRotationMatrix(  
    rotationMatrix, null, mGravity, mGeomagnetic);  
if (success) {  
    float orientationMatrix[] = new float[3];  
    // Returns the device's orientation given the rotationMatrix  
    SensorManager.getOrientation(rotationMatrix, orientationMatrix);  
    // Get the rotation, measured in radians, around the Z-axis  
    // Note: This assumes the device is held flat and parallel to the ground  
    float rotationInRadians = orientationMatrix[0];  
    // Convert from radians to degrees  
    mRotationInDegrees = Math.toDegrees(rotationInRadians);  
    // Request redraw  
    mCompassArrow.invalidate();  
    // Reset sensor event data arrays  
    mGravity = mGeomagnetic = null;  
    ...  
}
```



# Next Time

## Maps & Location

# Example Applications

SensorRawAccelerometer

SensorFilteredAccelerometer

SensorCompass