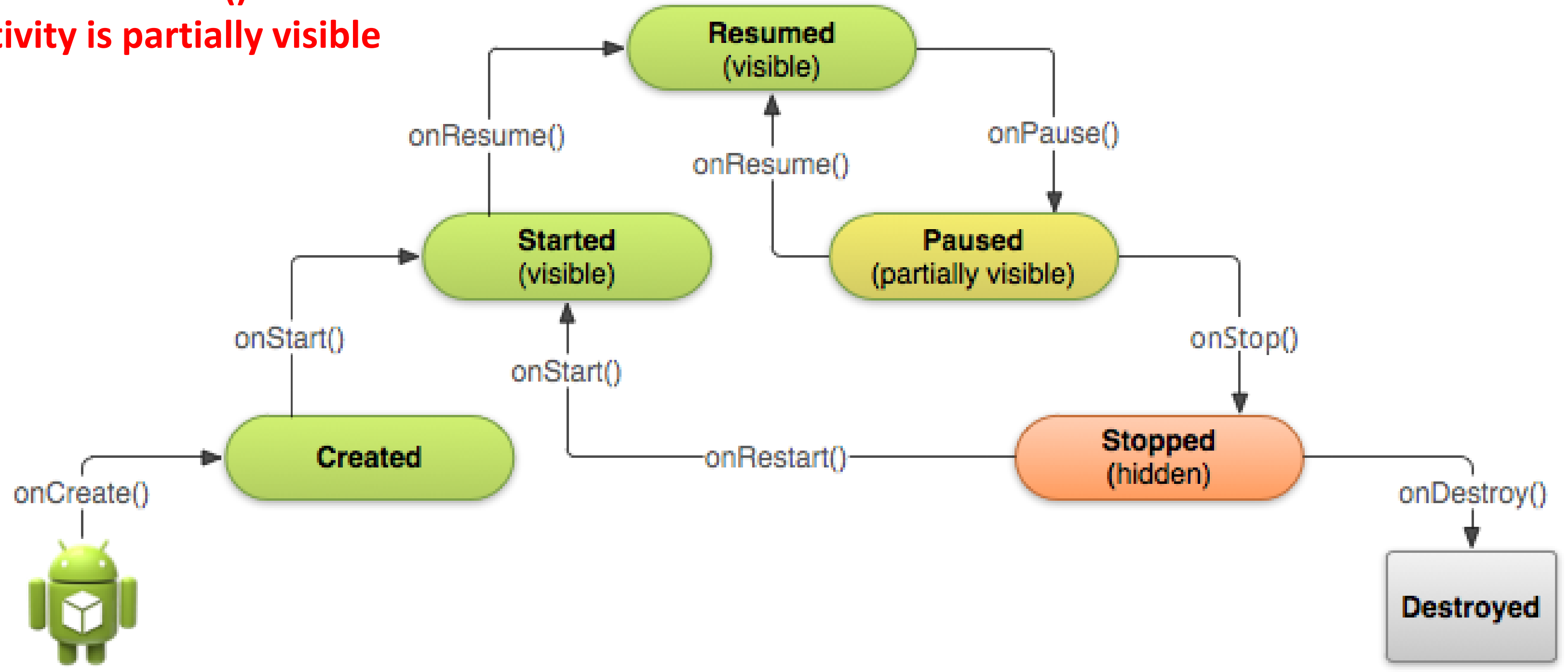
CSE 162 Mobile Computing

Lab 2 Sensor and Camera Programming

Starting an activity

- Activity starts
  - onCreate(), onStart(), onResume() are called in succession

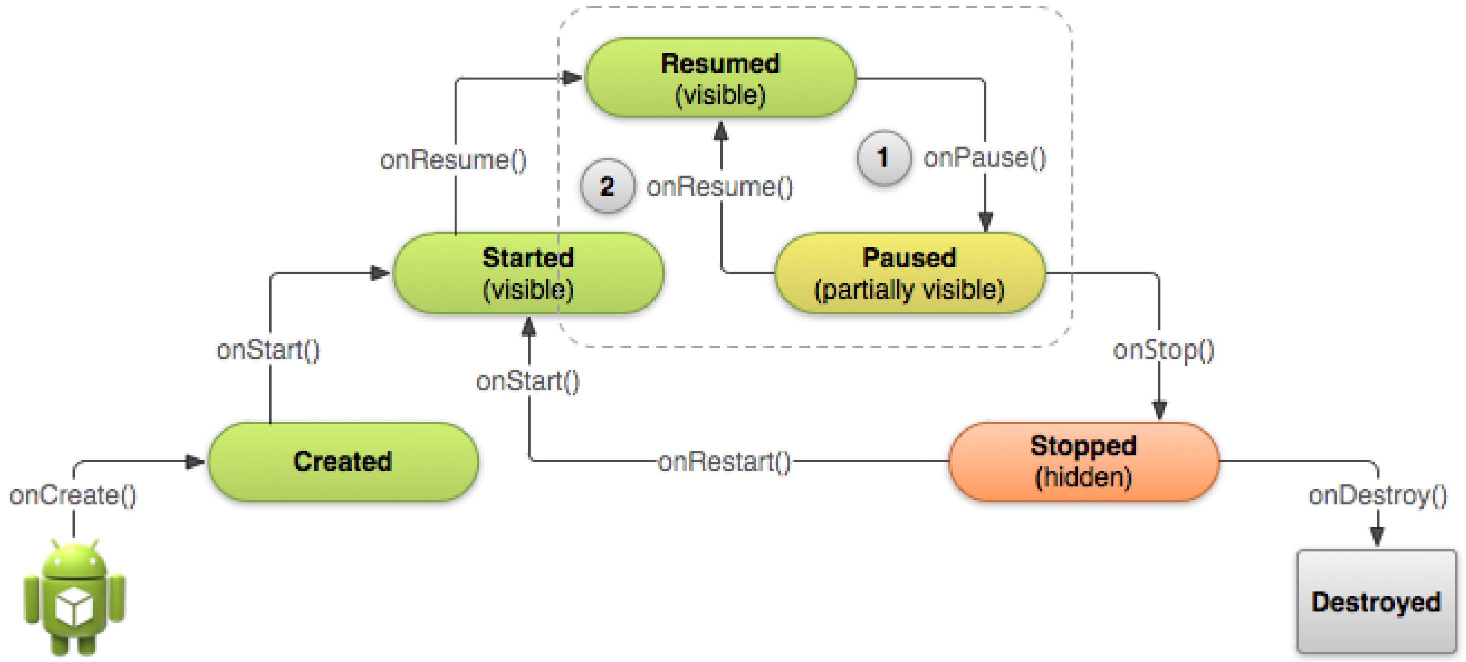
onStart()
Activity is partially visible



Pausing and Resuming an Activity

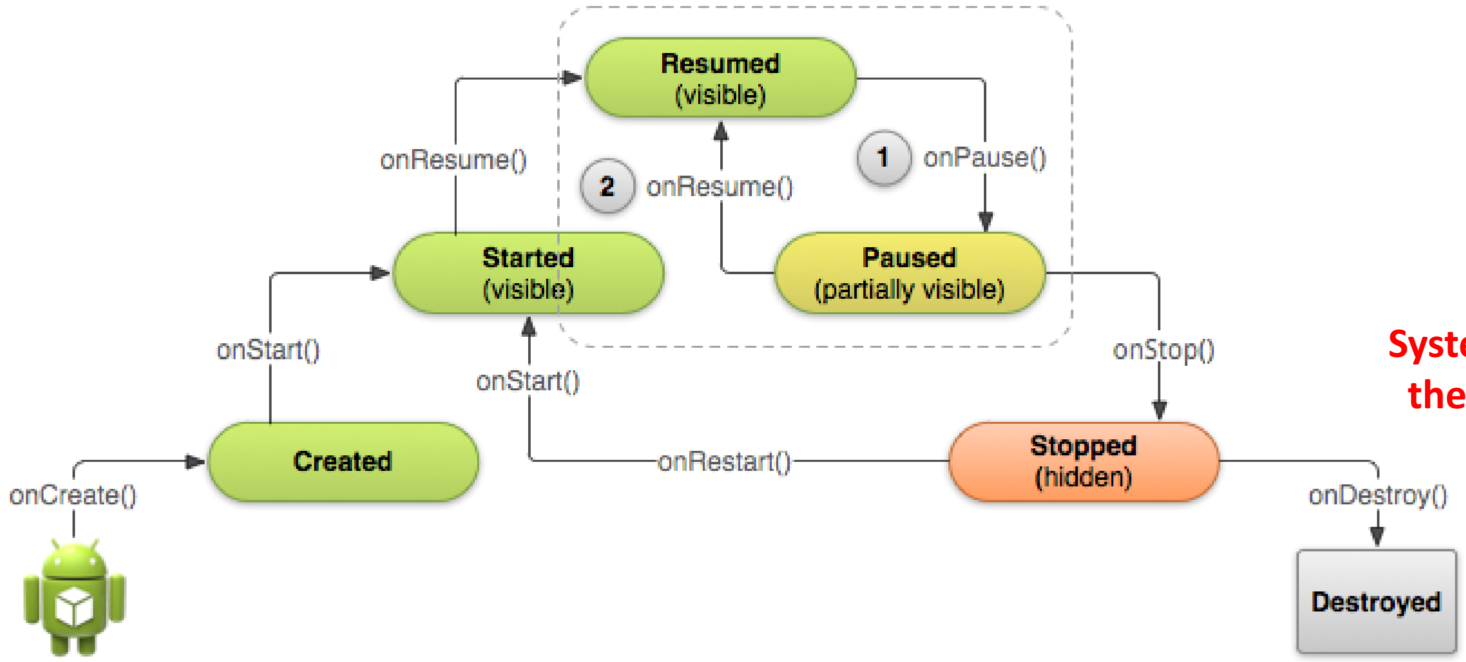
- How is pausing defined?
    - If an activity is partially visible in the background
    - Dialog boxes open up
    - Method called is onPause()

What should you do before an activity pauses
save state of the application?
release resources



Stopping and Starting an Activity

- How is stopping an activity defined?
  - If the activity is not visible
  - Press back or home button
  - Start a new activity or receive a phone call



System destroys the activity

Android Sensors Overview

- Android Sensors:
- MIC
- Camera
- Temperature
- Location (GPS or Network)
- Orientation
- Accelerometer
- Gravity
- Proximity
- Pressure
- Light
- https://developer.android.com/guide/topics/sensors/sensors_overview.html

Three Types of Sensors

- Motion sensors
  - measure acceleration forces and rotational forces along three axes.
  - accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- Environmental sensors
  - measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity.
  - barometers, photometers, and thermometers.
- Position sensors
  - measure the physical position of a device
  - orientation sensors and magnetometers.

Two types of sensors on the android platform

- Hardware sensors
  - Physical sensors present on the phone
  - Accelerometers, temperature, gyroscope

- Software sensors
  - Virtual sensors that are built on top of hardware sensors.
  - Orientation sensors --- accelerometer + gyroscope

Sensor Programming

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

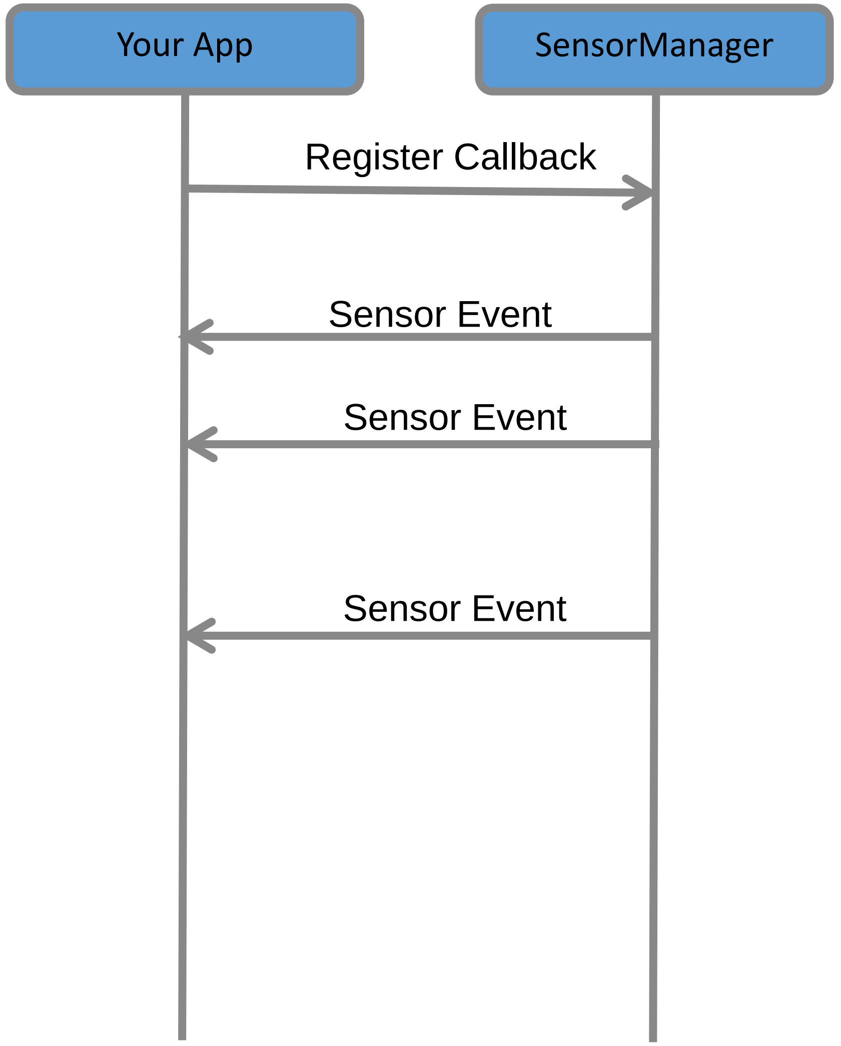
Async Callbacks

Android's sensors are controlled by external services and only send events when they choose to

•An app must register a callback to be notified of a sensor event

•Each sensor has a related XXXListener interface that your callback must implement

- E.g. LocationListener



Getting the Relevant System Service

- The non-media (e.g. not camera) sensors are managed by a variety of XXXXManager classes:
  - LocationManager (GPS)
  - SensorManager (accelerometer, gyro, proximity, light, temp)
- The first step in registering is to obtain a reference to the relevant manager
- Every Activity has a getSystemService() method that can be used to obtain a reference to the needed manager

```
public class MyActivity … {

  private SensorManager sensorManager_;

  public void onCreate(){
     …

     sensorManager_ = (SensorManager) getSystemService(SENSOR_SERVICE);
  }

}
```

Registering for Sensor Updates

- The SensorManager handles registrations for
  - Accelerometer, Temp, Light, Gyro
- In order for an object to receive updates from a sensor, it must implement the SensorEventListener interface
- Once the SensorManager is obtained, you must obtain a reference to the specific sensor you are interested in updates from
- The arguments passed into the registerListener method determine the sensor that you are connected to and the rate at which it will send you updates

```
public class MyActivity … implements SensorEventListener{
  private Sensor accelerometer_;
  private SensorManager sensorManager_;

  public void connectToAccelerometer() {
        sensorManager_ = (SensorManager)getSystemService(SENSOR_MANAGER);
        accelerometer_ =
sensorManager_.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorManager_.registerListener(this, accelerometer_,
                                 SensorManager.SENSOR_DELAY_NORMAL);



}
```

The SensorEventListener Interface

- Because there is one interface for multiple types of sensors, listening to multiple sensors requires switching on the type of event (or creating separate listener objects)
- Also forces registration at the same rate per listener
- Simple approach:

```
public class MyActivity … implements SensorEventListener{


          // Called when a registered sensor changes value
         @Override
         public void onSensorChanged(SensorEvent sensorEvent) {
                 if (sensorEvent.sensor.getType() ==
Sensor.TYPE_ACCELEROMETER) {
                         float xaccel = sensorEvent.values[0];
                         float yaccel = sensorEvent.values[1];
                         float zaccel = sensorEvent.values[2];
                 }
         }


         // Called when a registered sensor's accuracy changes
         @Override
         public void onAccuracyChanged(Sensor arg0, int arg1) {
                 // TODO Auto-generated method stub

         }
}
```

The SensorEventListener Interface

- Another approach for multiple sensors (probably better):

```
public class MyActivity … {

         private class AccelListener implements SensorEventListener {
                  public void onSensorChanged(SensorEvent sensorEvent) {
                           …
                  }
                  public void onAccuracyChanged(Sensor arg0, int arg1) {}
         }

         private class LightListener implements SensorEventListener {
                  public void onSensorChanged(SensorEvent sensorEvent) {
                           …
                  }
                  public void onAccuracyChanged(Sensor arg0, int arg1) {}
         }

         private SensorEventListener accelListener_ = new AccelListener();
         private SensorEventListener lightListener_ = new LightListener();

         …
         public void onResume(){
            …
          sensorManager_.registerListener(accelListener, accelerometer,
                                           SensorManager.SENSOR_DELAY_GAME);
          sensorManager_.registerListener(lightListener, lightsensor,
                                           SensorManager.SENSOR_DELAY_NORMAL);
         }
         public void onPause(){
           sensorManager_.unregisterListener(accelListener_);
           sensorManager_.unregisterListener(lightListener_);
```

Assignment 1: Detecting the Ambient Light

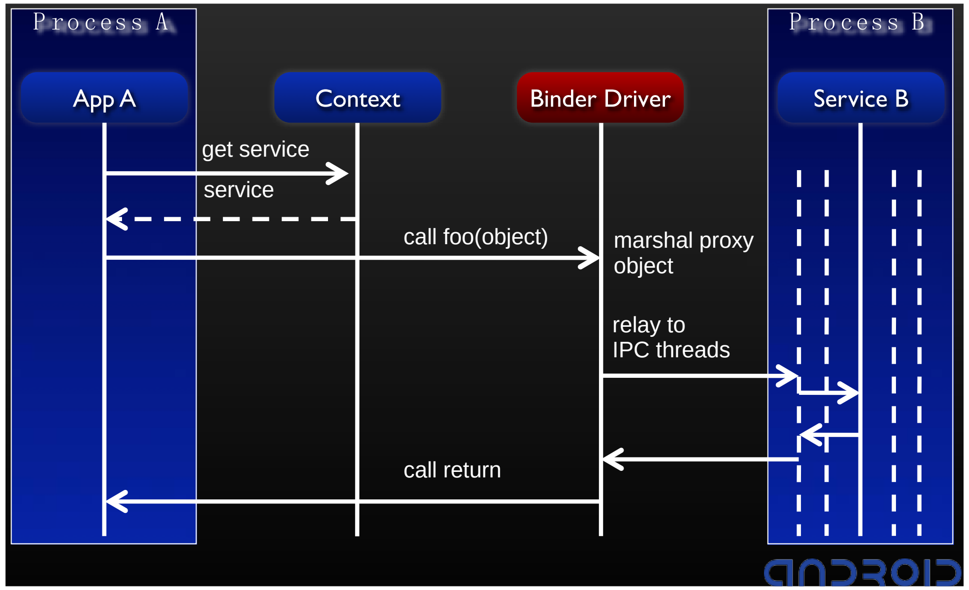
- Create an application that can display the light level of the environment in real-time.
- Using the light sensor.

Android System Services

- Each App runs in its own process
- Each Android system service, such as the SensorManager, also runs in its own thread

- This has important implications:

1. Communication with the system services is through IPC
2. The thread that delivers an event will be a special thread that is dedicated to processing incoming IPC calls
3. If you directly update the GUI from any thread other than the display thread, bad things happen
4. https://developer.android.com/reference/android/app/Service.html
5. https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html
6. https://developer.android.com/guide/components/processes-and-threads.html#Processes
7. https://developer.android.com/training/multiple-threads/communicate-ui.html



Android System Services

- Concurrency: Software that can do more than one thing at a time is known as *concurrent* software. The Java platform is designed from the ground up to support concurrent programming.
- Services, like other application objects, run in the main thread of their hosting process. This means that, if your service is going to do any CPU intensive (such as MP3 playback) or blocking (such as networking) operations, it should spawn its own thread in which to do that work.
- Main thread is also sometimes called the UI thread. It is in charge of dispatching events to the appropriate user interface widgets, including drawing events. It is also the thread in which your application interacts with components from the Android UI toolkit.
- Single thread model may make the application appears to hang. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog.
- Worker Threads. If you have operations to perform that are not instantaneous, you should make sure to do them in separate threads ("background" or "worker" threads).
- Only objects running on the UI thread have access to other objects on that thread. Because tasks that you run on a thread from a thread pool aren't running on your UI thread, they don't have access to UI objects. To move data from a background thread to the UI thread, use a Handler that's running on the UI thread.
- https://developer.android.com/reference/java/lang/Runnable.html

Functions are called in succession.

```
@Override
public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Get an instance of the sensor service
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        light = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

@Override
public final void onSensorChanged(SensorEvent event) {
    EditText field = findViewById(R.id.lightValue);
    field.setText(event.values[0] + " lux");
}
```

Assignment 2: Use Camera to take a photo.

- See
- http://developer.android.com/guide/topics/media/camera.html

- OPTION 1) Create Intent to existing Camera App--- just to take quick picture and get file returning to your app…camera picture controls not part of your app.
- https://developer.android.com/training/camera/photobasics.html

- OPTION 2) You can create your own camera controls inside your app and take a picture.

Some Classes involved…..

- Option 1: via Intent to existing Camera App
  - Intent An intent action type of MediaStore.ACTION_IMAGE_CAPTURE or MediaStore.ACTION_VIDEO_CAPTURE can be used to capture
- Option 2: via Your own controls and Interface HAS 2 Options
  - Camera This class is the primary API for controlling device cameras. This class is used to take pictures or videos when you are building a camera application..
  - SurfaceView This class is used to present a live camera preview to the user.
  - MediaRecorder This class is used to record video from the camera.
  - Newer **android.hardware.camera2**

Request Camera Permission

- We create a TakingPhoto project.
- To advertise that your application depends on having a camera, put a <user-feature> tag in your manifest file:

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

Take a Photo with the Camera App

- The Android way of delegating actions to other applications is to invoke an Intent that describes what you want done. This process involves three pieces: The Intent itself, a call to start the external Activity, and some code to handle the image data when focus returns to your activity.

```
static final int REQUEST_IMAGE_CAPTURE = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}
```

- resolveActivity() returns the first activity component that can handle the intent.
- If you call startActivityForResult() using an intent that no app can handle, your app will crash.
- So as long as the result is not null, it's safe to use the intent.t that no app can handle, your app will crash. So as long as the result is not null, it's safe to use the intent.

Get the Thumbnail

- We want to do some thing with the photo.
- The Android Camera application encodes the photo in the return Intent delivered to onActivityResult() as a small Bitmap in the extras, under the key "data". The following code retrieves this image and displays it in an ImageView.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        mImageView.setImageBitmap(imageBitmap);
    }
}
```

Save the Full-size Photo

- The Android Camera application saves a full-size photo if you give it a file to save into. You must provide a fully qualified file name where the camera app should save the photo.
- The write permission implicitly allows reading, so if you need to write to the external storage then you need to request permission:
- If you'd like the photos to remain private to your app only, you can instead use the directory provided by getExternalFilesDir(). On Android 4.3 and lower, writing to this directory also requires the WRITE_EXTERNAL_STORAGE permission. Beginning with Android 4.4, the permission is no longer required because the directory is not accessible by other apps, so you can declare the permission should be requested only on the lower versions of Android by adding the maxSdkVersion attribute:

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                     android:maxSdkVersion="18" />
    ...
</manifest>
```

Save the Full-size Photo

- Once you decide the directory for the file, you need to create a collision-resistant file name.
- You may wish also to save the path in a member variable for later use.

```
private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    File image = File.createTempFile(
        imageFileName,  /* prefix */
        ".jpg",         /* suffix */
        storageDir      /* directory */
    );

    // Save a file: path for use with ACTION_VIEW intents
    currentPhotoPath = image.getAbsolutePath();
    return image;
}
```

Save the Full-size Photo

- Modify the dispatchTakePictureIntent() method.

```
static final int REQUEST_TAKE_PHOTO = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
                                                  "com.example.android.fileprovider",
                                                  photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO);
        }
    }
}
```

Save the Full-size Photo

- Now, you need to configure the FileProvider. In your app's manifest, add a provider to your application:Now, you need to configure the FileProvider. In your app's manifest, add a provider to your application:

```
<application>
   ...
   <provider
        android:name="android.support.v4.content.FileProvider"
        android:authorities="com.example.android.fileprovider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/file_paths"></meta-data>
    </provider>
    ...
</application>
```

- Make sure that the authorities string matches the second argument to getUriForFile(Context, String, File). In the meta-data section of the provider definition, you can see that the provider expects eligible paths to be configured in a dedicated resource file, res/xml/file_paths.xml.

Create the resource file

- Create a xml directory under /res
- Create a file_path.xml file

```xml
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="my_images" path="" />
</paths>
```

Assignments: Light Sensor and Camera

- **Lab 2a**: Get and show the value of light senor.

  You can use emulators or real Android devices.

- **Lab 2b**: Take a picture by the camera.

  Use built-in camera to take a photo

  Return the full-size photo

  Display it on the screen

- **Scoring**

  Lab 2a: 50%

  Lab 2b: 50%