# Ubiquitous Computing

COMP H4025

Lecturer: Simon McLoughlin

Lecture 5

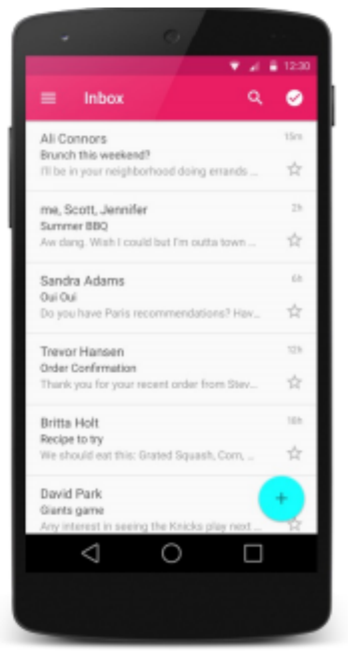
## Lecture Overview

This week:

- ListViews
- Displaying Bitmaps effectively
- Accessing the Camera

## ListView

*An ordered collection of selectable choices*



Ali Connors 15m
Brunch this weekend?
I'll be in your neighborhood doing errands ...

me, Scott, Jennifer 2h
Summer BBQ
Aw dang. Wish I could but I'm outta town ...

Sandra Adams 6h
Oui Oui
Do you have Paris recommendations? Hav...

Trevor Hansen 12h
Order Confirmation
Thank you for your recent order from Stev...

Britta Holt 18h
Recipe to try
We should eat this: Grated Squash, Corn, ...

David Park
Giants game
Any interest in seeing the Knicks play next ...

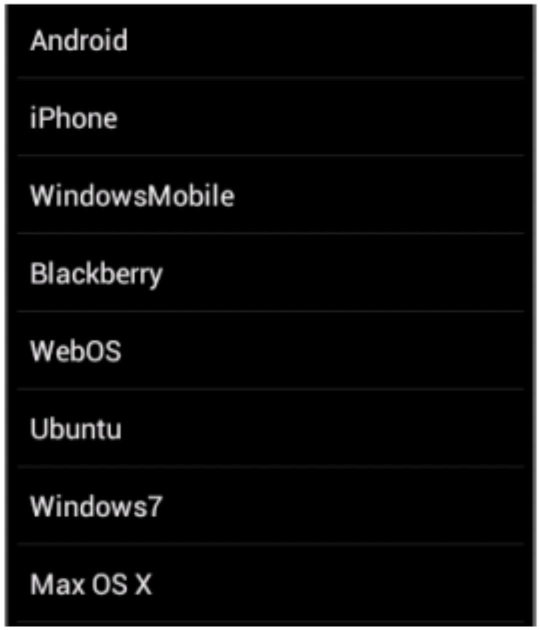
- key attributes in XML:

| | |
|---|---|
| `android:clickable="`***`bool`***`"` | set to false to disable the list |
| `android:id="@+id/`***`theID`***`"` | unique ID for use in Java code |
| `android:entries="@array/`***`array`***`"` | set of options to appear in the list (must match an array in `strings.xml`) |

## Static list

- **static list**: Content is fixed and known before the app runs.
  - Declare the list elements in the **strings.xml** resource file.

```
<!-- res/values/strings.xml -->
<resources>
    <string-array name="oses">
        <item>Android</item>
        <item>iPhone</item>
        ...
        <item>Max OS X</item>
    </string-array>
</resources>

<!-- res/layout/activity_main.xml -->
<ListView ... android:id="@+id/mylist"
    android:entries="@array/oses" />
```



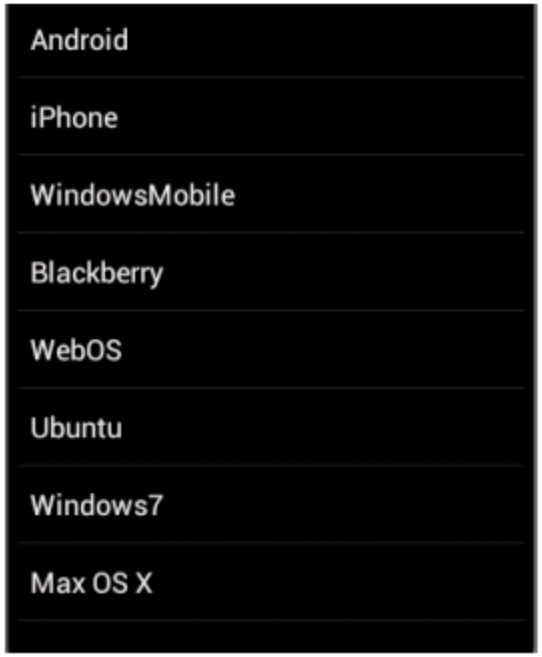
Android
iPhone
WindowsMobile
Blackberry
WebOS
Ubuntu
Windows7
Max OS X

## Dynamic list

- **dynamic list**: Content is read or generated as the program runs.
  - Comes from a data file, or from the internet, etc.
  - Must be set in the Java code.

  - Suppose we have the following file and want to make a list from it:

```
// res/raw/oses.txt
Android
iPhone
...
Max OS X
```



Android
iPhone
WindowsMobile
Blackberry
WebOS
Ubuntu
Windows7
Max OS X

## List Adapters

- **adapter**: Helps turn list data into list view items.
  - common adapters: `ArrayAdapter`, `CursorAdapter`

- Syntax for creating an adapter:

```
ArrayAdapter<String> name =
  new ArrayAdapter<String>(activity, layout, array);
```

  - the ***activity*** is usually `this`
  - the default ***layout*** for lists is `android.R.layout.simple_list_item_1`
  - get the ***array*** by reading your file or data source of choice
    (it can be an array like `String[]`, or a list like `ArrayList<String>`)

  - Once you have an adapter, you can attach it to your list by calling the `setAdapter` method of the `ListView` object in the Java code.

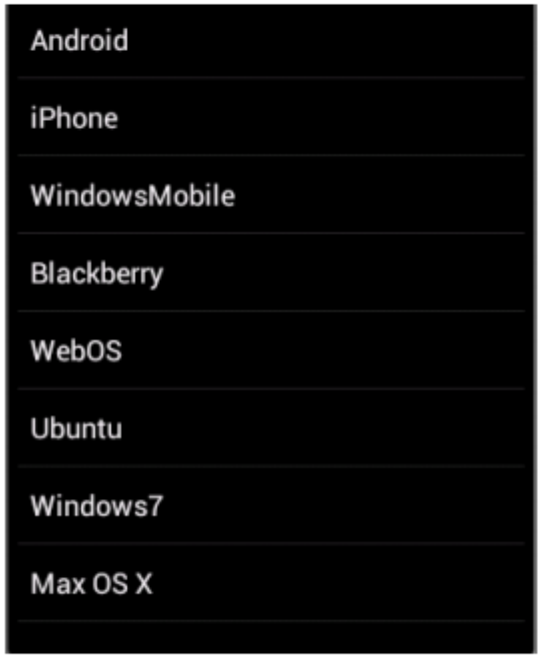
## List Adapters

```
ArrayList<String> myArray = ...;  // load data from file

ArrayAdapter<String> adapter =
  new ArrayAdapter<String>(
      this,
      android.R.layout.simple_list_item_1,
      myArray);

ListView list = (ListView) findViewById(R.id.mylist);
list.setAdapter(myAdapter);
```

## Handling List Events

- Unfortunately lists don't use a simple `onClick` event.
  - Several fancier GUI widgets use other kinds of events.
  - The event listeners must be attached in the Java code, not in the XML.
  - Understanding how to attach these event listeners requires the use of Java **anonymous inner classes**.

- **anonymous inner class**: A shorthand syntax for declaring a small class without giving it an explicit name.
  - The class can be made to extend a given super class or implement a given interface.
  - Typically the class is declared and a single object of it is constructed and used all at once.



## Handling List Events

```
<!-- activity_main.xml -->
~~<Button ... android:onClick="mybuttonOnClick" />~~
<Button ... android:id="@+id/mybutton" />

// MainActivity.java
~~public void mybuttonOnClick() { ... }~~
Button button = (Button) findViewById(R.id.mybutton);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // code to run when the button gets clicked
    }
});

// this was the required style for event listeners
// in older versions of Android :-/
```

## Handling List Events

- List views respond to the following events:
  - **setOnItemClickListener**(AdapterView.OnItemClickListener)

    Listener for when an item in the list has been clicked.

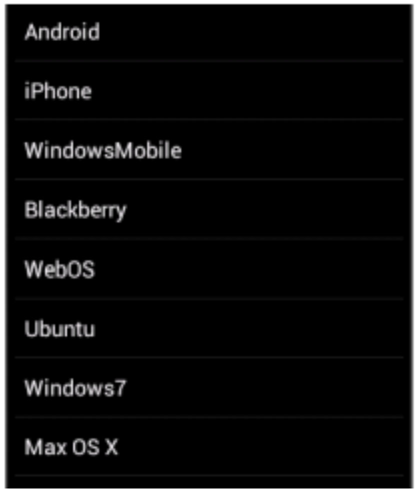
  - **setOnItemLongClickListener**(AdapterView.OnItemLongClickListener)

    Listener for when an item in the list has been clicked and held.

  - **setOnItemSelectedListener**(AdapterView.OnItemSelectedListener)

    Listener for when an item in the list has been selected.

- Others:
  - onDrag, onFocusChanged, onHover, onKey, onScroll, onTouch, ...

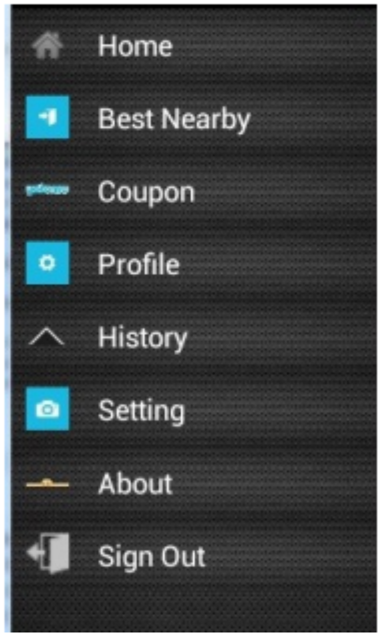


## List Event Listener

```
ListView list = (ListView) findViewById(R.id.id);
list.setOnItemClickListener(
    new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> list,
                                View row,
                                int index,
                                long rowID) {
            // code to run when user clicks that item
            ...
        }
    }
);
```

## Custom Lists

- If you want your list to look different than the default appearance (of just a text string for each line), you must:
  - Write a short **layout XML file** describing the layout for each row.
  - Write a **subclass of `ArrayAdapter`** that overrides the **`getView`** method to describe what view must be returned for each row.



## Custom List Layout

```
<!-- res/layout/mylistlayout.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ... android:orientation="horizontal">
     <ImageView ... android:id="@+id/list_row_image"
          android:layout_width="100dp"
          android:layout_height="100dp"
          android:src="@drawable/smiley" />

     <TextView ... android:id="@+id/list_row_text"
          android:textStyle="bold"
          android:textSize="22dp"
          android:text=""
          android:background="#336699" />
</LinearLayout>
```

## Custom List Layout

```java
// MyAdapter.java
public class MyAdapter extends ArrayAdapter<String> {
    private int layoutResourceId;
    private List<String> data;

    public MyAdapter(Context context, int layoutId, List<String> list) {
        super(context, layoutResourceId, data);
        layoutResourceId = layoutId;
        data = list;
    }

    @Override
    public View getView(int index, View row, ViewGroup parent) {
        row = getLayoutInflater().inflate(layoutResourceId, parent, false);
        TextView text = (TextView) row.findViewById(R.id.list_row_text);
        text.setText(data.get(index));
        return row;
    }
}
```

## Displaying Bitmaps Efficiently

- Bitmaps are just structured arrays of numbers. Each number has a location in the map. If the number is a zero or a one then it is a bit that has a coordinate in a map, or a bitmap.
- Normally the numbers will not be single bits but perhaps a byte or if the image is colour then three bytes per pixel, one for each of the RGB channels.
- If you consider the amount of memory required for a Bitmap, it can be a lot. For example, the camera on the Galaxy Nexus takes photos up to 2592x1936 pixels (5 megapixels). If the bitmap configuration used is ARGB_8888 (the default from the Android 2.3 onward) then loading this image into memory takes about 19MB of memory (2592*1936*4 bytes), immediately exhausting the per-app limit on some devices (which can be as little as 16MB).
- So its important that you can decode or resize large bitmaps without exceeding the per application memory limit. Normally you will want to resize a bitmap to the same size as devices display or a component within it.

## Read Bitmap Dimensions and Type

- The first thing you want to do is read the bitmap dimension and type to decide on how to deal with it.
- The BitmapFactory class provides several decoding methods (decodeByteArray(), decodeFile(),decodeResource(), etc.) for creating a Bitmap from various sources.

These methods attempt to allocate memory for the constructed bitmap and therefore can easily result in an OutOfMemory exception. Each type of decode method has additional signatures that let you specify decoding options via the BitmapFactory.Options class. Setting theinJustDecodeBounds property to true while decoding avoids memory allocation, returning null for the bitmap object but setting outWidth, outHeight and outMimeType. This technique allows you to read the dimensions and type of the image data prior to construction (and memory allocation) of the bitmap.

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(getResources(), R.id.myimage, options);
int imageHeight = options.outHeight;
int imageWidth = options.outWidth;
String imageType = options.outMimeType;
```

## Loading a smaller version

• Now that the image dimensions are known, they can be used to decide if the full image should be loaded into memory or if a subsampled version should be loaded instead.
• Here are some factors to consider:
- Estimated memory usage of loading the full image in memory.
- Amount of memory you are willing to commit to loading this image given any other memory requirements of your application.
- Dimensions of the target ImageView or UI component that the image is to be loaded into.
- Screen size and density of the current device.

• For example, it's not worth loading a 1024x768 pixel image into memory if it will eventually be displayed in a 128x96 pixel thumbnail in an ImageView.
• To tell the decoder to subsample the image, loading a smaller version into memory, set inSampleSize to true in your BitmapFactory.Options object.
• For example, an image with resolution 2048x1536 that is decoded with an inSampleSize of 4 produces a bitmap of approximately 512x384.
• Loading this into memory uses 0.75MB rather than 12MB for the full image (assuming a bitmap configuration of ARGB_8888). Here's a method to calculate a sample size value that is a power of two based on a target width and height:

## Loading a smaller version

```java
public static int calculateInSampleSize(
            BitmapFactory.Options options, int reqWidth, int reqHeight) {
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if (height > reqHeight || width > reqWidth) {

        final int halfHeight = height / 2;
        final int halfWidth = width / 2;

        // Calculate the largest inSampleSize value that is a power of 2 and keeps both
        // height and width larger than the requested height and width.
        while ((halfHeight / inSampleSize) > reqHeight
                && (halfWidth / inSampleSize) > reqWidth) {
            inSampleSize *= 2;
        }
    }

    return inSampleSize;
}
```

## Loading a smaller version

- To use this method, first decode with inJustDecodeBounds set to true, pass the options through and then decode again using the new inSampleSize value and inJustDecodeBounds set to false:

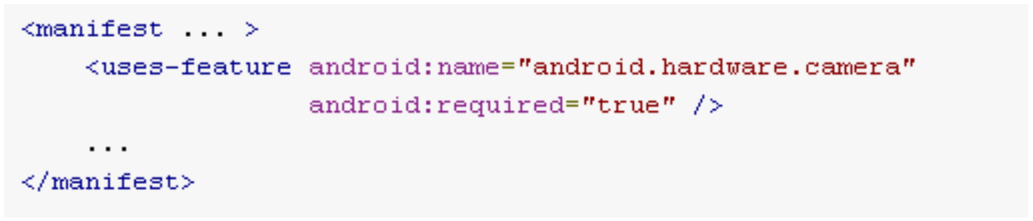
```
public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,
        int reqWidth, int reqHeight) {

    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeResource(res, resId, options);

    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeResource(res, resId, options);
}
```

- This method makes it easy to load a bitmap of arbitrarily large size into an ImageView that displays a 100x100 pixel thumbnail, as shown in the following example code:

```
mImageView.setImageBitmap(
    decodeSampledBitmapFromResource(getResources(), R.id.myimage, 100, 100));
```

## Accessing the Device Camera

- If an essential function of your application is taking pictures, then restrict its visibility on Google Play to devices that have a camera. To advertise that your application depends on having a camera, put a <uses-feature> tag in your manifest file:



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

- There are two ways to get photos or pictures into your app, you can start and Intent that actions the native camera app to do it and return the picture to your app or you can use the Camera API and write the code yourself

## Using Existing Camera Apps

- A quick way to enable taking pictures or videos in your application without a lot of extra code is to use an Intent to invoke an existing Android camera application.

- A camera intent makes a request to capture a picture or video clip through an existing camera app and then returns control back to your application.

```
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);
    }
}
```

- 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 anImageView.

```
@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);
    }
}
```

## Building a Camera App

The general steps for creating a custom camera interface for your application are as follows:

**Detect and Access Camera** - Create code to check for the existence of cameras and request access.
**Create a Preview Class** - Create a camera preview class that extends SurfaceView and implements theSurfaceHolder interface. This class previews the live images from the camera.
**Build a Preview Layout** - Once you have the camera preview class, create a view layout that incorporates the preview and the user interface controls you want.
**Setup Listeners for Capture** - Connect listeners for your interface controls to start image or video capture in response to user actions, such as pressing a button.
**Capture and Save Files** - Setup the code for capturing pictures or videos and saving the output.
**Release the Camera** - After using the camera, your application must properly release it for use by other applications.

## Detecting and Accessing the Camera

•If your application does not specifically require a camera using a manifest declaration, you should check to see if a camera is available at runtime. To perform this check, use the PackageManager.hasSystemFeature()method, as shown in the example code below:

```
/** Check if this device has a camera */
private boolean checkCameraHardware(Context context) {
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        // this device has a camera
        return true;
    } else {
        // no camera on this device
        return false;
    }
}
```

• Android devices can have multiple cameras, for example a back-facing camera for photography and a front-facing camera for video calls. Android 2.3 (API Level 9) and later allows you to check the number of cameras available on a device using the Camera.getNumberOfCameras() method.

## Accessing the Camera

• If you have determined that the device on which your application is running has a camera, you must request to access it by getting an instance of Camera (unless you are using an intent to access the camera).
To access the primary camera, use the Camera.open() method and be sure to catch any exceptions, as shown in the code below:

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance(){
    Camera c = null;
    try {
        c = Camera.open(); // attempt to get a Camera instance
    }
    catch (Exception e){
        // Camera is not available (in use or does not exist)
    }
    return c; // returns null if camera is unavailable
}
```

• On devices running Android 2.3 (API Level 9) or higher, you can access specific cameras usingCamera.open(int). The example code above will access the first, back-facing camera on a device with more than one camera.

## Checking Camera Features

- Once you obtain access to a camera, you can get further information about its capabilities using theCamera.getParameters() method and checking the returned Camera.Parameters object for supported capabilities. When using API Level 9 or higher, use the Camera.getCameraInfo() to determine if a camera is on the front or back of the device, and the orientation of the image.

- You can also change camera features using the Camera.setParameters() method.

- Typical camera features might be frame rate, exposure time, flash, scene mode, resolution, focal length etc.

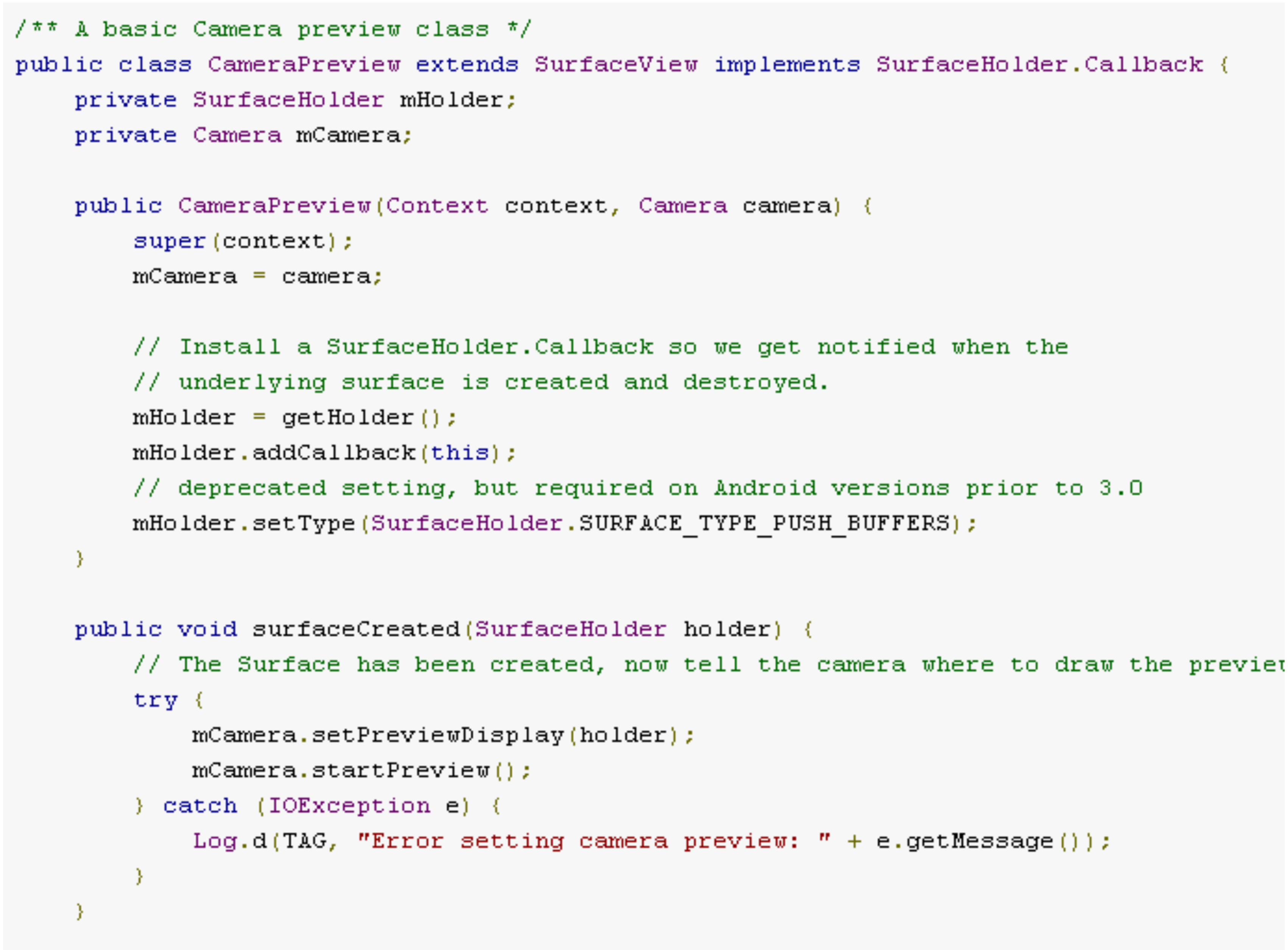
## Creating the Preview class

- For users to effectively take pictures or video, they must be able to see what the device camera sees. A camera preview class is a SurfaceView that can display the live image data coming from a camera, so users can frame and capture a picture or video.

- The following example code demonstrates how to create a basic camera preview class that can be included in a View layout.

- This class implements SurfaceHolder.Callback in order to capture the callback events for creating and destroying the view, which are needed for assigning the camera preview input.

## Creating the Preview class



```
/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
    private SurfaceHolder mHolder;
    private Camera mCamera;

    public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;

        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder) {
        // The Surface has been created, now tell the camera where to draw the previe
        try {
            mCamera.setPreviewDisplay(holder);
            mCamera.startPreview();
        } catch (IOException e) {
            Log.d(TAG, "Error setting camera preview: " + e.getMessage());
        }
    }
```

## Creating the Preview class

```
public void surfaceDestroyed(SurfaceHolder holder) {
    // empty. Take care of releasing the Camera preview in your activity.
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
    // If your preview can change or rotate, take care of those events here.
    // Make sure to stop the preview before resizing or reformatting it.

    if (mHolder.getSurface() == null){
      // preview surface does not exist
      return;
    }

    // stop preview before making changes
    try {
        mCamera.stopPreview();
    } catch (Exception e){
      // ignore: tried to stop a non-existent preview
    }

    // set preview size and make any resize, rotate or
    // reformatting changes here

    // start preview with new settings
    try {
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();

    } catch (Exception e){
        Log.d(TAG, "Error starting camera preview: " + e.getMessage());
    }
}
```

## Placing preview in a layout

- A camera preview class, such as the example shown in the previous slide, must be placed in the layout of an activity along with other user interface controls for taking a picture or video.

- The following layout code provides a very basic view that can be used to display a camera preview. In this example, the FrameLayout element is meant to be the container for the camera preview class.

- This layout type is used so that additional picture information or controls can be overlayed on the live camera preview images.

## Placing preview in a layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <FrameLayout
    android:id="@+id/camera_preview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    />

  <Button
    android:id="@+id/button_c
    android:text="Capture"
    android:layout_width="wra
    android:layout_height="wr
    android:layout_gravity="c
    />
</LinearLayout>
```

```java
public class CameraActivity extends Activity {

    private Camera mCamera;
    private CameraPreview mPreview;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Create an instance of Camera
        mCamera = getCameraInstance();

        // Create our Preview view and set it as the content of our activity.
        mPreview = new CameraPreview(this, mCamera);
        FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
        preview.addView(mPreview);
    }
}
```

## Placing preview in a layout

• On most devices, the default orientation of the camera preview is landscape. This example layout specifies a horizontal (landscape) layout and the code below fixes the orientation of the application to landscape.

• For simplicity in rendering a camera preview, you should change your application's preview activity orientation to landscape by adding the following to your manifest.

```
<activity android:name=".CameraActivity"
          android:label="@string/app_name"

          android:screenOrientation="landscape">
          <!-- configure this activity to use landscape orientation -->

          <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

## Capturing pictures

• Once you have built a preview class and a view layout in which to display it, you are ready to start capturing images with your application.

• In your application code, you must set up listeners for your user interface controls to respond to a user action by taking a picture.

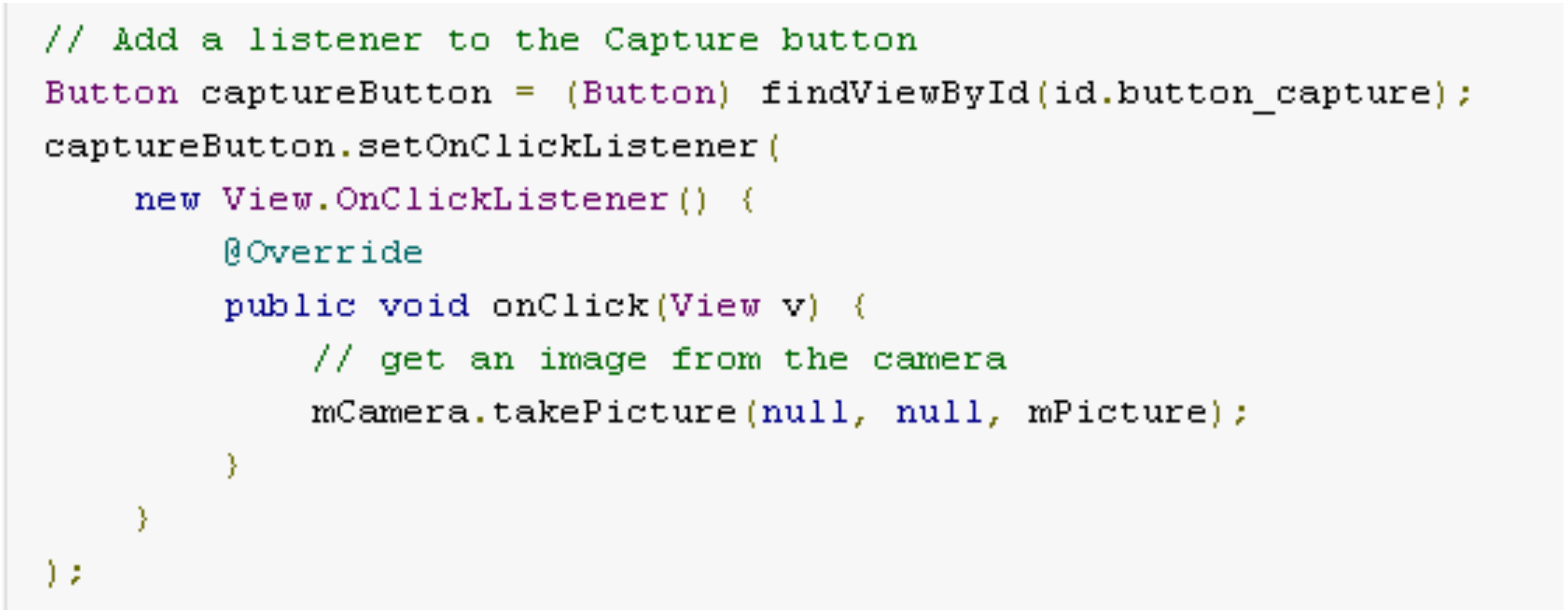
• In order to retrieve a picture, use the Camera.takePicture() method. This method takes three parameters which receive data from the camera. In order to receive data in a JPEG format, you must implement an Camera.PictureCallback interface to receive the image data and write it to a file.

• The following code shows a basic implementation of the Camera.PictureCallback interface to save an image received from the camera.

## Capturing pictures

```
private PictureCallback mPicture = new PictureCallback() {

    @Override
    public void onPictureTaken(byte[] data, Camera camera) {

        File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
        if (pictureFile == null){
            Log.d(TAG, "Error creating media file, check storage permissions: " +
                e.getMessage());
            return;
        }

        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        } catch (FileNotFoundException e) {
            Log.d(TAG, "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.d(TAG, "Error accessing file: " + e.getMessage());
        }
    }
};
```

## Capturing pictures



```
// Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
    new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // get an image from the camera
            mCamera.takePicture(null, null, mPicture);
        }
    }
);
```

- You must be particularly careful to release the camera object when your application stops using it, and as soon as your application is paused (Activity.onPause()).
- If your application does not properly release the camera, all subsequent attempts to access the camera, including those by your own application, will fail and may cause your or other applications to be shut down.
- To release an instance of the Camera object, use the Camera.release() method.

## Camera Features

• Android supports a wide array of camera features you can control with your camera application, such as picture format, flash mode, focus settings, and many more.

• You can check the availabilty of camera features by getting an instance of a camera's parameters object, and checking the relevant methods. The following code sample shows you how to obtain a Camera.Parameters object and check if the camera supports the autofocus feature:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();

List<String> focusModes = params.getSupportedFocusModes();
if (focusModes.contains(Camera.Parameters.FOCUS_MODE_AUTO)) {
  // Autofocus mode is supported
}
```

## Setting Camera Features

• Most camera features are activated and controlled using a Camera.Parameters object. You obtain this object by first getting an instance of the Camera object, calling the getParameters() method, changing the returned parameter object and then setting it back into the camera object, as demonstrated in the following example code:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();
// set the focus mode
params.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
// set Camera parameters
mCamera.setParameters(params);
```

• This technique works for nearly all camera features, and most parameters can be changed at any time after you have obtained an instance of the Camera object. Changes to parameters are typically visible to the user immediately in the application’s camera preview.
• On the software side, parameter changes may take several frames to actually take effect as the camera hardware processes the new instructions and then sends updated image data.