In the last chapter, we looked at how we can leverage Android's built-in Camera application to provide a ready-made photo capture component in any other application. While this provides a standard interface to the end user and is straightforward for us the programmers, it doesn’t provide us with much in the way of flexibility. For instance, if we wanted our photo capture application to support time-lapse photography, we couldn’t easily do that using the built-in application.

Fortunately, Android doesn’t limit us to just using the built-in applications for accessing the hardware camera. We have as much access to the underlying hardware and available methods as the Camera application itself, which allows us to use those capabilities in any type of application we would like.

In this chapter, we’ll explore building a photo-taking application utilizing the underlying Camera class and learn how to exploit the capabilities we are given. We’ll go through the steps required to build a few different applications:

- A straightforward point and shoot photo app
- A countdown-style timer
- A time-lapse photo-taking application

**Using the Camera Class**

The Camera class in Android is what we use to access the camera hardware on the device. It allows us to actually capture an image, and through its nested Camera.Parameters class, we can change set various attributes, such as whether the flash should be activated and what value the white balance should be set to.

Camera Permissions

In order to use the Camera class to capture an image, we need to specify in our AndroidManifest.xml file that we require the CAMERA permission.

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

Preview Surface

Also before we can get started using the camera, we need to create some type of Surface for the Camera to draw viewfinder or preview images on. A Surface is an abstract class in Android representing a place to draw graphics or images. One straightforward way to provide a drawing Surface is to use the SurfaceView class. SurfaceView is a concrete class providing a Surface within a standard View.

To specify a SurfaceView in our layout, we simply use the `<SurfaceView />` element within any normal layout XML. Here is a basic layout that just implements a SurfaceView within a LinearLayout for a camera preview.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <SurfaceView android:id="@+id/CameraView" android:layout_width="fill_parent"
        android:layout_height="fill_parent"></SurfaceView>
</LinearLayout>
```

In our code, for the purposes of using this SurfaceView with the Camera class, we'll need to add a SurfaceHolder to the mix. The SurfaceHolder class can act as a monitor on our Surface, giving us an interface through callbacks to let us know when the Surface is created, destroyed, or changed. The SurfaceView class conveniently gives us a method, `getHolder`, to obtain a SurfaceHolder for its Surface.

Here is a snippet of code that accesses the SurfaceView as declared in the layout XML and obtains a SurfaceHolder from it. It also sets the Surface to be a “push” type of Surface, which means that the drawing buffers are maintained external to the Surface itself. In this case, the buffers are managed by the Camera class. A “push” type of Surface is required for the Camera preview.

```java
SurfaceView cameraView = (CameraView) this.findViewById(R.id.CameraView);
SurfaceHolder surfaceHolder = cameraView.getHolder();
surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Additionally, we'll want to implement SurfaceHolder.Callable in our activity. This allows our activity to be notified when the Surface is created, when it changes and when it is destroyed. To implement the Callback, we'll add the following methods.

```java
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {}
public void surfaceCreated(SurfaceHolder holder) {}
public void surfaceDestroyed(SurfaceHolder holder) {}
```