Assignment C-3

Aim:

To develop an android snapshot application.

Problem Statement:

Design suitable assignment for Mobile Programming [Optional: to take a snapshot using mobile camera.]

Theory:

Request Camera Permission:

To advertise that an application depends on having a camera, a <uses-feature> tag is required in the manifest file:

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

If your application uses, but does not require a camera in order to function, instead set android:required to false.

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);
    }
}
```

Notice that the startActivityForResult() method is protected by a condition that calls resolveActivity(), which returns the first activity component that can handle the intent. Performing this check is important because 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.

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.

Generally, any photos that the user captures with the device camera should be saved on the device in the public external storage so they are accessible by all apps. The proper directory for shared photos is provided by getExternalStoragePublicDirectory(), with the DIRECTORY_PICTURES argument. Because the directory provided by this method is shared among all apps, reading and writing to it requires the READ_EXTERNAL_STORAGE

and WRITE_EXTERNAL_STORAGE permissions, respectively. The write permission implicitly allows reading, so if you need to write to the external storage then you need to request only one permission:

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

Add the Photo to a Gallery:

When you create a photo through an intent, you should know where your image is located, because you said where to save it in the first place. For everyone else, perhaps the easiest way to make your photo accessible is to make it accessible from the system's Media Provider.

The following example method demonstrates how to invoke the system's media scanner to add your photo to the Media Provider's database, making it available in the Android Gallery application and to other apps.

```
private void galleryAddPic() {
    Intent mediaScanIntent = new Intent(
                        Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);

    File f = new File(mCurrentPhotoPath);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    this.sendBroadcast(mediaScanIntent);
}
```

The methods onAccuracyChanged() and onSensorChanged() are being declared because the class is implementing the SensorEventListener interface, which allows the definition of the sensor callbacks. That means that it's possible to execute a code every time the values from a sensor change or when the accuracy of the information returned by a sensor changes.

Decode a Scaled Image:

Managing multiple full-sized images can be tricky with limited memory. If you find your application running out of memory after displaying just a few images, you can dramatically reduce the amount of dynamic heap used by expanding the JPEG into a memory array that's already scaled to match the size of the destination view. The following example method demonstrates this technique.

```
private void setPic() {
    // Get the dimensions of the View
    int targetW = mImageView.getWidth();
    int targetH = mImageView.getHeight();

    // Get the dimensions of the bitmap
    BitmapFactory.Options bmOptions =
                                    new BitmapFactory.Options();
    bmOptions.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);
    int photoW = bmOptions.outWidth;
    int photoH = bmOptions.outHeight;

    // Determine how much to scale down the image
    int scaleFactor = Math.min(photoW/targetW, photoH/targetH);

    // Decode the image file into a Bitmap sized to
                                                //fill the View
    bmOptions.inJustDecodeBounds = false;
    bmOptions.inSampleSize = scaleFactor;
    bmOptions.inPurgeable = true;

    Bitmap bitmap = BitmapFactory.decodeFile(
                                   mCurrentPhotoPath, bmOptions);
    mImageView.setImageBitmap(bitmap);
}
```

Conclusion:

Thus, we have studied how to build an android application to use the camera application for taking snapshots.

Mathematical Modeling:

Let S be the system that represents the Snapshot Application.

Initially,

$$\mathbf{S} = \{ \phi \}$$

Let,

$$\mathbf{S} = \{ \mathbf{I, O, F} \}$$

Where,
I - Represents Input set
O - Represents Output set
F - Represents Function set

Input set - I:

Filename of the image to be captured.

Output set - O:

Captured image using the camera of the mobile.

Function set - F:

$$\mathbf{F} = \{ F_1, F_2, F_3 \}$$

F_1 - Represents the onClick function for capture button.
F_1(E) –> { O_1, O_2 ..... O_n }
Where,

- E: Event Handler.
- O_i: ith operation.

F_2 - Represents the function for creating the application directory.
F_2() –> { D }
Where,

- D: Directory created for Snapshot application.

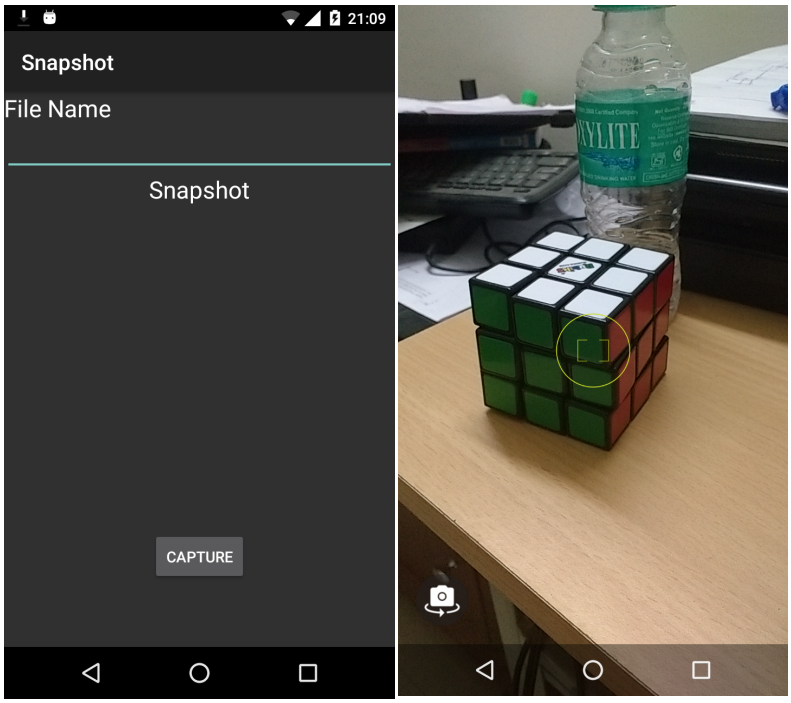
F_3 - Represents the function for scaling the captured image.
F_3(I) –> { I_s }
Where,

- I: Original Image.
- I_s: Scaled image.

Finally,

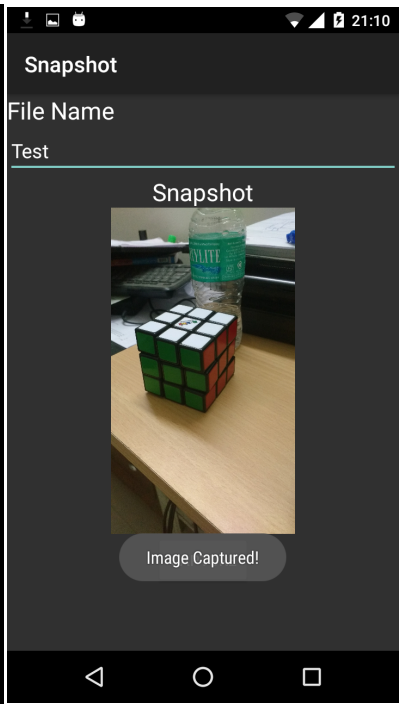
S ={ I, O, F }

Output:



(a) Output (b) Output

(c) Output



(d) Output