Implicit Intents Part Two-16

In this assignment we are going to expand on our implicit intents work from the previous lab. We will add the ability to share a photo with a friend and also integrate the camera. We will use the camera to attach an image to our photo record.

1. Create a new repository on GitHub and add your instructor as a collaborator.

2. Open your Soo Greyhounds app in Android Studio. Ensure you have submitted your previous lab. Remove the remote repository that is pointing to your previous lab and add the remote to your new repository. Perform a push to ensure everything is setup properly.

3. We are going to add the ability to share a photo with friends. To do this we will implement some code to build a message when a user sends a photo to their friends. First, let's add some strings we will need to do this. In strings.xml, add the strings shown below.

```
<resources>
  …
  <string name="photo_person">Choose Person</string>
  <string name="photo_details">%1$s! Check out this photo from the Soo Greyhounds app: %2$s.</string>
  <string name="photo_details_subject">Soo Greyhounds Photo Link</string>
  <string name="share_photo">Share Photo</string>
  <string name="send_photo">Send Photo</string>
</resources>
```

4. Next, add a button in your activity_photo_detail.xml that will be used to share the photo. Be sure to give the button an id of "share_photo_button". Also, be sure to use the share_photo string from the previous step as the text for the button.

5. In PhotoDetailsActivity, add a method that creates four strings and then pieces them together and returns a complete report. Note that there are two DateFormat classes: android.text.format.DateFormat and java.text.DateFormat. Use android.text.format.DateFormat.

```
  …
@Override
  public void onActivityResult(int requestCode, int resultCode, Intent data) {
    …
  }

  private String getPhotoDetails() {
    String details = getString(R.string.photo_details, mPhoto.getTitle(), mPhoto.getURL());
    return details;
  }
```

6. Now we can implement an implicit intent. In onCreate(…), get a reference to the share photo button and set a listener on it. Within the listener's implementation, create an implicit intent and pass it into startActivity(Intent).

```
…
private Button mPersonButton;
private Button mShareButton;
...

@Override
protected void onCreate(Bundle savedInstanceState) {
   ...
   mShareButton = (Button) findViewById(R.id.share_photo_button);
   mShareButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
         Intent i = new Intent(Intent.ACTION_SEND);
         i.setType("text/plain");
         i.putExtra(Intent.EXTRA_TEXT, getPhotoDetails());
         i.putExtra(Intent.EXTRA_SUBJECT, getString(R.string.photo_details_subject));
         startActivity(i);
      }
   });
}
```

7. Run SooGreyhounds and press the share photo button. Because this intent will likely match many activities on the device, you will probably see a list of activities presented in a chooser.

8. In PhotoDetailActivity.java, create your own chooser to display the activities that respond to your implicit intent.

```
mShareButton = (Button) findViewById(R.id.share_photo_button);
mShareButton.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
      Intent i = new Intent(Intent.ACTION_SEND);
      i.setType("text/plain");
      i.putExtra(Intent.EXTRA_TEXT, getPhotoDetails());
      i.putExtra(Intent.EXTRA_SUBJECT, getString(R.string.photo_details_subject));
      i = Intent.createChooser(i, getString(R.string.send_photo));
      startActivity(i);
   }
});
```

9. Run SooGreyhounds and press the share photo button. As long as you have more than one activity that can handle your intent, you will be offered a list to choose from.

10. Now, let's tackle the photo and camera work. We will need to add an image view and button to handle our photos. Open activity_photo_detail.xml and add:
    a) An ImageView for the picture, give it an id of "photo"
    b) An ImageButton to take a picture, give it an id of "camera_button"

11. Now, to respond to presses on your ImageButton and to control the content of your ImageView, you need instance variables referring to each of them. Call findViewById(int) as usual on your inflated activity_photo_detail.xml to find your new views and wire them up.

```
…
private Button mPersonButton;
private Button mShareButton;
private ImageButton mPhotoButton;
private ImageView mPhotoView;
...

@Override
protected void onCreate(Bundle savedInstanceState) {
      ...
      mPersonButton = findViewById(R.id.choose_person_button);
      mPhotoButton = (ImageButton) findViewById(R.id.camera_button);
      mPhotoView = (ImageView) findViewById(R.id.photo);
      …
 }
```

12. Let's setup our FileProvider. The first step is to declare FileProvider as a ContentProvider hooked up to a specific authority. This is done by adding a content provider declaration to your AndroidManifest.xml.

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

       <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
 </activity>
 <provider
   android:name="androidx.core.content.FileProvider"
   android:authorities="com.soogreyhounds.soogreyhoundsmobile.fileprovider"
   android:exported="false"
   android:grantUriPermissions="true">
</provider>
…
```

13. Now that you have told Android where your FileProvider is, you also need to tell your FileProvider which files it is exposing. This bit of configuration is done with an extra XML resource file. Right-click your app/res folder in the project tool window and select New → Android resource file. For Resource type, select XML, and then enter “files” for the name.

14. Open xml/files.xml, switch to the Text tab, and replace its contents with the following:

```
<paths>
   <files-path name="photos" path="."/>
</paths>
```

15. Now hook up files.xml to your FileProvider by adding a meta-data tag in your AndroidManifest.xml.

```
 <provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.soogreyhounds.soogreyhoundsmobile.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
       android:name="android.support.FILE_PROVIDER_PATHS"
       android:resource="@xml/files"/>
 </provider>
```

16. While you are in the manifest, let’s add the permission required by some Android versions to allow us to read photos taken from the camera.

```
<manifest …
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<application…
```

17. Time to give your pictures a place to live locally. First, add a method to Photo.java to get a well-known filename.

```
  …
  public void setPerson(String person) {
      mPerson = person;
   }

   public String getPhotoFilename() {
      return "IMG_" + getUUID() + ".jpg";
   }
}
```

18. Photo.getPhotoFilename() will not know what folder the photo will be stored in. However, the filename will be unique, since it is based on the Photo's UUID. Next, find where the photos should live. PhotoStorage is responsible for everything related to persisting data in SooGreyhounds, so it is a logical owner for this idea. Add a getPhotoFile(Photo) method to PhotoStorage that provides a complete local filepath for Photo's image.

```
public class PhotoStorage {
   ...
   public void addPhoto(Photo p) {
      ContentValues values = getContentValues(p);
      mDatabase.insert(SooGreyhoundsDBSchema.PhotoTable.NAME, null, values);
   }

   public File getPhotoFile(Photo photo) {
      File filesDir = mContext.getFilesDir();
      return new File(filesDir, photo.getPhotoFilename());
   }
   …
}
```

19. Let's integrate the camera. Start by storing the photo in a variable within PhotoDetailActivity.java.

```
…
private Photo mPhoto;
private File mPhotoFile;
...
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   …
   mPhoto = new Photo();

   if (getIntent().hasExtra(EXTRA_UUID)) {
         …
         if (mPhoto.getPerson() != null) {
            mPersonButton.setText(mPhoto.getPerson());
         }

         mPhotoFile = PhotoStorage.get(this).getPhotoFile(mPhoto);
      }
   …
}
```

20. Write an implicit intent to ask for a new picture to be taken into the location saved in mPhotoFile. Add code to ensure that the button is disabled if there is no camera app or if there is no location at which to save the photo.

```
private static final int REQUEST_CONTACT = 1;
private static final int REQUEST_PHOTO = 2;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ...
   mShareButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
         …
      }
    });

    final Intent captureImage = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    boolean canTakePhoto = mPhotoFile != null && captureImage.resolveActivity(packageManager) != null;

    if (!canTakePhoto) {
       mPhotoButton.setVisibility(View.GONE);
    }

    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
           Uri uri = FileProvider.getUriForFile(getBaseContext(),
                 "com.soogreyhounds.soogreyhoundsmobile.fileprovider",
                 mPhotoFile);
           captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);

           List<ResolveInfo> cameraActivities = getBaseContext()
                 .getPackageManager().queryIntentActivities(captureImage,
                       PackageManager.MATCH_DEFAULT_ONLY);

           for (ResolveInfo activity : cameraActivities) {
              getBaseContext().grantUriPermission(activity.activityInfo.packageName,
                    uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
           }

           startActivityForResult(captureImage, REQUEST_PHOTO);
        }
    });
}
```

21. Run SooGreyhounds, open an existing image (if you do not have one, create one, save it, then open it after saving) and press the camera button to run your camera app.

22. Create a new class called PictureUtils (create a new file called PictureUtils.java) for your new method and add a static method to it called getScaledBitmap(String, int, int).

```
public class PictureUtils {
   public static Bitmap getScaledBitmap(String path, int destWidth, int destHeight) {
      // Read in the dimensions of the image on disk
      BitmapFactory.Options options = new BitmapFactory.Options();
      options.inJustDecodeBounds = true;
      BitmapFactory.decodeFile(path, options);

      float srcWidth = options.outWidth;
      float srcHeight = options.outHeight;

      // Figure out how much to scale down by
      int inSampleSize = 1;
      if (srcHeight > destHeight || srcWidth > destWidth) {
         float heightScale = srcHeight / destHeight;
         float widthScale = srcWidth / destWidth;

         inSampleSize = Math.round(heightScale > widthScale ? heightScale :
               widthScale);
      }

      options = new BitmapFactory.Options();
      options.inSampleSize = inSampleSize;

      // Read in and create final bitmap
      return BitmapFactory.decodeFile(path, options);
   }
}
```

23. Write another static method called getScaledBitmap(String, Activity) to scale a Bitmap for a particular Activity's size.

```
public class PictureUtils {
   public static Bitmap getScaledBitmap(String path, Activity activity) {
      Point size = new Point();
      activity.getWindowManager().getDefaultDisplay().getSize(size);

      return getScaledBitmap(path, size.x, size.y);
   }
   …
}
```

24. Next, to load this Bitmap into your ImageView, add a method to PhotoDetialActivity.java to update mPhotoView.

```
    private String getPhotoDetails() {
        …
    }

    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
        } else {
            Bitmap bitmap = PictureUtils.getScaledBitmap(mPhotoFile.getPath(), this);
            mPhotoView.setImageBitmap(bitmap);
        }
    }
```

25. Then call that method from inside onCreate (…) and onActivityResult(…) in PhotoDetailActivity.java.

```
    …
    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ...
            startActivityForResult(captureImage, REQUEST_PHOTO);
        }
    });
    updatePhotoView();
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }
    if (requestCode == REQUEST_CONTACT && data != null) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
        Uri uri = FileProvider.getUriForFile(this,
            "com.soogreyhounds.soogreyhoundsmobile.fileprovider",
            mPhotoFile);

        this.revokeUriPermission(uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION);

        updatePhotoView();
    }
}
```

26. Run SooGreyhounds again, and you should see your image displayed in the thumbnail view.

27. Your camera implementation works great now. One more task remains: Tell potential users about it. When your app uses a feature like the camera – or near-field communication, or any other feature that may vary from device to device – it is strongly recommended that you tell Android about it. This allows other apps (like the Google Play Store) to refuse to install your app if it uses a feature the device does not support. To declare that you use the camera, add a <uses-feature> tag to your AndroidManifest.xml.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.soogreyhounds.soogreyhoundsmobile">

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