Implicit Intents Part Onep-15

In this assignment we are going to enhance our app to allow adding and editing of photos. We are also going to use implicit intents to enable picking a person from our contacts to attach to a photo. The user will choose a person from whatever contacts app is installed on the device.

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. Let's start by finishing what we started last lab and build in the ability to add and edit photos. First we will remove the manually added photo as we will now get data from the user through a form when we add a photo. Open MainActivity.java and remove the code from our add photo menu button press.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  …
 case R.id.add_photo:
      Photo photo = new Photo();
      photo.setUUID("234243-dsfsa-23sdf");
      photo.setTitle("A photo");
      photo.setURL("https://www.test.com/Test.jpg");
      photo.setNote("This is a note");

      PhotoStorage.get(getBaseContext()).addPhoto(photo);
      updateList();
      return true;
      …
  }
```

4. We are now going to setup our add photo button to open the photo detail screen. This is where users will enter details on the photo. We also want our list of photos on MainActivity to refresh when the user comes back. To do this we will open our activity expecting a result, that way when it closes we will be notified and can refresh the list. Add a request code constant to use when we open the photo detail screen.

```
public class MainActivity extends AppCompatActivity {
   private final int REQUEST_PHOTO = 1;
   …
}
```

5. Now that we have that, let's setup the code to open our screen.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
 …
    case R.id.add_photo:
      Intent addPhotoIntent = new Intent(this, PhotoDetailActivity.class);
      startActivityForResult(addPhotoIntent, REQUEST_PHOTO);
      return true;
      …
}
```

6. Before we move onto our photo detail screen, let's setup the code that will refresh our list of photos when the user comes back from the photo detail screen. In MainActivity.java, add the onActivityResult() override and refresh the list.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (resultCode == Activity.RESULT_OK) {
    switch (requestCode) {
      case REQUEST_PHOTO:
        updateList();
      break;
    }
  }
}
```

7. We are now setup to open the photo detail screen, and refresh our list when the user comes back. Let's create some form fields on our photo detail screen so we can add different photos. We are going to have a button to save a photo, so let's add a string for that button.

```
<resources>
  …
  <string name="save_photo">Save Photo</string>
</resources>
```

8. Open the activity_photo_detail.xml layout file and add the following:
   a) An EditText widget (if you are using the graphic editor, these are called Plain Text) with the id photo_uuid
   b) An EditText widget with the id photo_title
   c) An EditText widget with the id photo_url
   d) A button widget with the id save_photo (be sure to set the text of this button to the string you created in the previous step)

9. Now let's hook up our widgets so we can use them. First, setup class variables for the EditText widgets so we can use them in our code that saves the photo.

```
public class PhotoDetailActivity extends AppCompatActivity {
   private EditText mUUIDEditText;
   private EditText mTitleEditText;
   private EditText mURLEditText;
   …
```

10. We are also going to need a variable to hold the photo being added/edited so let's add a class variable for that.

```
public class PhotoDetailActivity extends AppCompatActivity {
   private Photo mPhoto;
```

11. Next, get the widgets from our layout and store them in the variables (we'll also add a local variable for our button as we do not need that to be at the class level).

```
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_photo_detail);

      mUUIDEditText = findViewById(R.id.photo_uuid);
      mTitleEditText = findViewById(R.id.photo_title);
      mURLEditText = findViewById(R.id.photo_url);

      Button saveButton = findViewById(R.id.save_photo);
      …
```

12. Now hook up the button and implement the code to save the photo.

```
Button saveButton = findViewById(R.id.save_photo);
saveButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mUUIDEditText.getText().toString().equals("")) {
            Toast.makeText(v.getContext(), "Please enter a UUID", Toast.LENGTH_LONG).show();
            return;
        }
        mPhoto.setUUID(mUUIDEditText.getText().toString());
        mPhoto.setTitle(mTitleEditText.getText().toString());
        mPhoto.setURL(mURLEditText.getText().toString());

        PhotoStorage.get(v.getContext()).addPhoto(mPhoto);
        setResult(Activity.RESULT_OK);
        finish();
    }
});
```

13. Next we will setup the ability edit a photo. This will entail opening a photo from the list in MainActivity.java and loading the data for that photo. We are going to use an extra to pass the photo's UUID from our list into our photo detail screen. Before we go over to MainActivity.java, let's first add an extra constant we can use for the UUID.

```
public class PhotoDetailActivity extends AppCompatActivity {
    public static String EXTRA_UUID = "com.soogreyhounds.soogreyhoundsmobile.photo.uuid";
```

14. Open MainActivity.java and implement the code to open the photo that is pressed (here is where we set the extra to the UUID so we can load the photo on the photo detail screen).

```
private class PhotoHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
   private Photo mPhoto;
   …
   public PhotoHolder(View itemView) {
      super(itemView);
      itemView.setOnClickListener(this);
      mTitleTextView = (TextView) itemView;
   }

   public void bindPhoto(Photo photo) {
      mPhoto = photo;
      mTitleTextView.setText(photo.getTitle());
   }

   @Override
   public void onClick(View v) {
      Intent editPhotoIntent = new Intent(v.getContext(), PhotoDetailActivity.class);
      editPhotoIntent.putExtra(PhotoDetailActivity.EXTRA_UUID, mPhoto.getUUID());
      startActivityForResult(editPhotoIntent, REQUEST_PHOTO);
   }
}
```

15. Now let's go update our photo detail screen to handle not only the adding of photos, but also the updating. Open PhotoDetailActivity.java and implement code to load a photo if is an edit instead of an add (we will also set a boolean variable to indicate if it is and edit or not, this will be used when we save to determine if it should be an insert or update). Also, if it is an edit we will prevent the changing of the UUID as that is used to determine which is to be updated in the database.

```
public class PhotoDetailActivity extends AppCompatActivity {
   …
   private boolean mEditing;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_photo_detail);

      mUUIDEditText = findViewById(R.id.photo_uuid);
      mTitleEditText = findViewById(R.id.photo_title);
      mURLEditText = findViewById(R.id.photo_url);

      mEditing = false;

      mPhoto = new Photo();

      if (getIntent().hasExtra(EXTRA_UUID)) {
         mEditing = true;
         String uuid = getIntent().getStringExtra(EXTRA_UUID);
         mPhoto = PhotoStorage.get(this).getPhoto(uuid);

         mUUIDEditText.setText(uuid);
         mUUIDEditText.setEnabled(false);
         mTitleEditText.setText(mPhoto.getTitle());
         mURLEditText.setText(mPhoto.getURL());       }
      …
   }
   …
}
```

16. Now update the save code to check and see if we should update or add.

```
Button saveButton = findViewById(R.id.save_photo);
 saveButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
       if (mUUIDEditText.getText().toString().equals("")) {
          Toast.makeText(v.getContext(), "Please enter a UUID", Toast.LENGTH_LONG).show();
          return;
       }

       mPhoto.setUUID(mUUIDEditText.getText().toString());
       mPhoto.setTitle(mTitleEditText.getText().toString());
       mPhoto.setURL(mURLEditText.getText().toString());

       if (mEditing) {
          PhotoStorage.get(v.getContext()).updatePhoto(mPhoto);
       } else {
          PhotoStorage.get(v.getContext()).addPhoto(mPhoto);
       }

       setResult(Activity.RESULT_OK);
       finish();
    }
 });
```

17. Let's run our app to ensure there are no problems. Add a photo, then click it in the list and edit it. Ensure the changes show when you come back from the photo detail screen.

18. We will now start working on the pick a contact code. We will update the PhotoDetailActivity layout to include a button for selecting a person. First, add a string that the button will display as it's text.

```
<resources>
   …
   <string name="photo_person">Choose Person</string>
</resources>
```

19. In layout/activity_photo_detail.xml, add a button using the photo_person string you created in the previous step, give it an id choose_person_button

20. Next, open Photo.java and add a new member variable to give Photo a field that will hold the name of a person attached to the photo.

```
public class Photo {
   ...
   private String mNote;
   private String mPerson;
   ...
   public String getPerson() {
      return mPerson;
   }

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

21. Now you need to add an additional field to your photo database. First, add a person column to SooGreyhoundsDBSchema.

```
public class SooGreyhoundsDBSchema {
   public static final class PhotoTable {
      public static final String NAME = "photos";

      public static final class Cols {
         …
         public static final String PERSON = "person";
      }
   }
}
```

22. Also, add the column in SooGreyhoundsDBHelper. (don't forget the quotation mark and comma after SooGreyhoundsDBSchema.PhotoTable.Cols.NOTE +.)

```
   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL("create table " + SooGreyhoundsDBSchema.PhotoTable.NAME + "(" +
         " _id integer primary key autoincrement, " +
         SooGreyhoundsDBSchema.PhotoTable.Cols.UUID + ", " +
         SooGreyhoundsDBSchema.PhotoTable.Cols.TITLE + ", " +
         SooGreyhoundsDBSchema.PhotoTable.Cols.URL + ", " +
         SooGreyhoundsDBSchema.PhotoTable.Cols.NOTE + ", " +
         SooGreyhoundsDBSchema.PhotoTable.Cols.PERSON +
         ")"
      );
   }
```

23. Next, write to the new column in PhotoStorage.getContentValues(Photo).

```
private static ContentValues getContentValues(Photo photo) {
    ContentValues values = new ContentValues();
    values.put(SooGreyhoundsDBSchema.PhotoTable.Cols.UUID, photo.getUUID());
    values.put(SooGreyhoundsDBSchema.PhotoTable.Cols.TITLE, photo.getTitle());
    values.put(SooGreyhoundsDBSchema.PhotoTable.Cols.URL, photo.getURL());
    values.put(SooGreyhoundsDBSchema.PhotoTable.Cols.NOTE, photo.getNote());
    values.put(SooGreyhoundsDBSchema.PhotoTable.Cols.PERSON, photo.getPerson());
    return values;
}
```

24. Now read from it in PhotoCursorWrapper.

```
public Photo getPhoto() {
    String uuid = getString(getColumnIndex(SooGreyhoundsDBSchema.PhotoTable.Cols.UUID));
    String title = getString(getColumnIndex(SooGreyhoundsDBSchema.PhotoTable.Cols.TITLE));
    String url = getString(getColumnIndex(SooGreyhoundsDBSchema.PhotoTable.Cols.URL));
    String note = getString(getColumnIndex(SooGreyhoundsDBSchema.PhotoTable.Cols.NOTE));
    String person = getString(getColumnIndex(SooGreyhoundsDBSchema.PhotoTable.Cols.PERSON));

    Photo photo = new Photo();
    photo.setUUID(uuid);
    photo.setTitle(title);
    photo.setURL(url);
    photo.setNote(note);
    photo.setPerson(person);

    return photo;
}
```

25. If SooGreyhounds is already installed on your device, your existing database will not have the person column, and your new onCreate(SQLiteDatabase) will not be run to add the new column, either. As we said earlier, the easiest solution is to wipe out your old database so Android will create your database again using your new onCreate(…) code (this happens a lot in app development.) First, uninstall the SooGreyhounds app by opening the app launcher screen and dragging the SooGreyhounds icon to the top of the screen. All your sandbox storage will get blown away, along with the out-of-date database schema, as part of the uninstall process. Next, run SooGreyhounds from Android Studio. A new database will be created with the new column as part of the app installation process.

26. Alright, let's get to the implicit intents! First, open PhotoDetailActivity.java and add a constant for the request code and a class variable for the button we will use to select a person from our contacts.

```
public class PhotoDetailActivity extends AppCompatActivity {
    private static final int REQUEST_CONTACT = 1;
    ...
    private EditText mURLEditText;

    private Button mPersonButton;
    …

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        …
        mURLEditText = findViewById(R.id.photo_url);

        mPersonButton = findViewById(R.id.choose_person_button);
        …
    }
}
```

27. At the end of onCreate(…), get a reference to the button and set a listener on it. Within the listener's implementation, create the implicit intent and pass it into startActivityForResult(…). Also, once a person is assigned. You will be using pickContact one more time in a bit, which is why you put it outside mPersonButton's OnClickListener

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        final Intent pickContact = new Intent(Intent.ACTION_PICK,
                ContactsContract.Contacts.CONTENT_URI);

        mPersonButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                startActivityForResult(pickContact, REQUEST_CONTACT);
            }
        });
    }
```

28. Given we will be displaying the name of the person attached to the photo on the button, we should load the name (if one exists) when the user edits. Do this in the piece of code where we handle loading data into the EditTexts when a photo is being edited.

```
@Override
   protected void onCreate(Bundle savedInstanceState) {
      …

      if (getIntent().hasExtra(EXTRA_UUID)) {
         mEditing = true;
         String uuid = getIntent().getStringExtra(EXTRA_UUID);
         mPhoto = PhotoStorage.get(this).getPhoto(uuid);

         mUUIDEditText.setText(uuid);
         mUUIDEditText.setEnabled(false);
         mTitleEditText.setText(mPhoto.getTitle());
         mURLEditText.setText(mPhoto.getURL());

         if (mPhoto.getPerson() != null) {
            mPersonButton.setText(mPhoto.getPerson());
         }
      }
     …
}
```

29. Now let's retrieve the selected contact's name from the contacts application by implementing onActivityResult(…)in PhotoDetailActivity.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (resultCode != Activity.RESULT_OK) {
    return;
  }

  if (requestCode == REQUEST_CONTACT && data != null) {
    Uri contactUri = data.getData();
    // Specify which fields you want your query to return
    // values for
    String[] queryFields = new String[] {
        ContactsContract.Contacts.DISPLAY_NAME
    };

    // Perform your query - the contactUri is like a "where" clause here
    Cursor c = getContentResolver().query(contactUri, queryFields, null, null, null);

    try {
      // Double-check that you actually got results
      if (c.getCount() == 0) {
        return;
      }

      // Pull out the first column of the first row of data -
      // that is your person's name
      c.moveToFirst();
      String person = c.getString(0);
      mPhoto.setPerson(person);
      mPersonButton.setText(person);
    } finally {
      c.close();
    }
  }
}
```

30. Go ahead and run your app and test your contacts app integration.

31. Some devices or users may not have a contacts app, and if the OS cannot find a matching activity, then the app will crash. The fix is to check with part of the OS called the PackageManager first. Do this in onCreateView(…).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    …
  final Intent pickContact = new Intent(Intent.ACTION_PICK,
  ContactsContract.Contacts.CONTENT_URI);

   mPersonButton = findViewById(R.id.choose_person_button);
   mPersonButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
         startActivityForResult(pickContact, REQUEST_CONTACT);
      }
   });

   PackageManager packageManager = getPackageManager();
   if (packageManager.resolveActivity(pickContact,
         PackageManager.MATCH_DEFAULT_ONLY) == null) {
      mPersonButton.setEnabled(false);
   }
  …
}
```

32. Run your app again and ensure everything works.