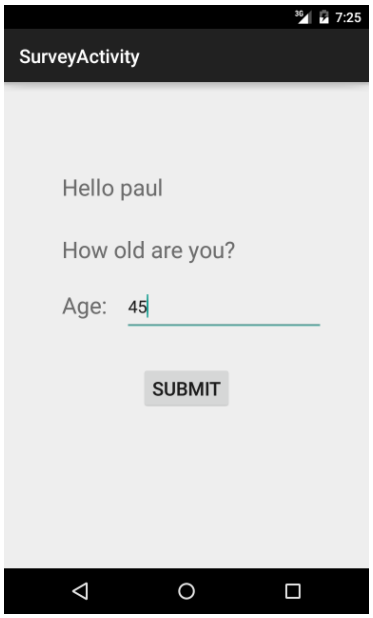
CSCD 372: Android Development
Lab 1: Activities, Intents, and Notifications

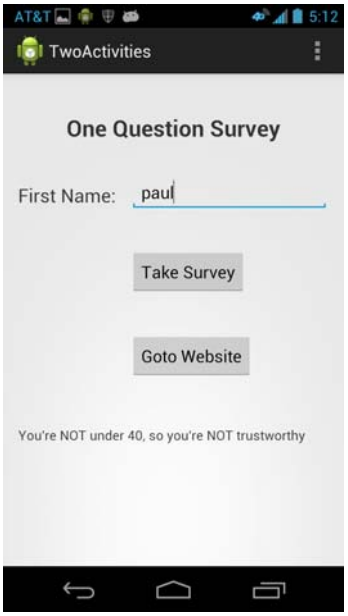
Lab Overview

The purpose of this lab is for you to gain some familiarity with Activities and Intents, including Implicit Intents. You will learn how to pass data between activities, and you will also do some runtime investigation of Activity lifecycles by instrumenting at least one lifecycle callback using system notifications.

The end product of this lab will be a simple program with two activities. Typing a name (e.g., "paul") in the first activity (called TwoActivities below) and pressing Take Survey leads to the second Activity (called SurveyActivity below). Note that the name entered in the first Activity is passed to the second, where it is used in the greeting.

Entering an Age in the second activity and pressing Submit sends that age value back to the first activity, where it is used to format the text message at the bottom of the screen, as shown in the 3rd screenshot. If the age is 40 or more, this message should say that you are NOT trustworthy, as shown. Otherwise it should say that you are trustworthy, also as shown except without the two NOT words.





Activity 1 Activity 2 Activity 1 again

Pressing the Goto Website button will use an implicit Intent to launch a browser with the following website loaded: https://sites.google.com/site/pschimpf99/

The first activity will also have the ability to handle the implicit ACTION_SEND Intent on mimeType "text/plain" data, by displaying the text in the bottom text field. A convenient way to test this is by selecting Share Page from the context menu of your web browser. Android should

give you a list of apps that can handle this, and if you select your app from the list, the URL should show up in the bottom text field, which is initialized to "(results will appear here)".

When all that is done you will instrument the onCreate() method of the main (first) activity, as detailed in Task 4 below. Answer the questions on the last page when you have that working.

Following are some suggestions and hints for how to proceed. You need not follow this order of development, it is simply a suggestion. You should, however, at least read through the steps because they provide some additional details on the app's behavior.

Task 1: Create an empty project shell and modify the main activity layout

1. Start with the Empty or Blank wizards and make the appropriate changes either way as covered in module 2. You can see from the screenshot above that I called my app "TwoActivities", but you should call yours something like "phschimpfLab1" where you substitute your initials and last name for "phschimpf".
2. Modify the contents of the main activity so it contains the widgets shown above. You can use the default Relative Layout. I'm not particularly concerned about the constraints on the layout for this lab, as long as it looks reasonably close to the screenshots above.
3. Add a handler for the Take Survey button using any of the methods covered in module 2. Have it display a Toast or something else simple for now.
4. Add a handler for the Goto Website button. In response, create an implicit Intent to go to https://sites.google.com/site/pschimpf99/
   This is done by instantiating a Uri object using the Uri.parse() method, passing it the desired URL as a string. Then instantiate an Intent of type Intent.ACTION_VIEW by passing that Uri to the Intent constructor. Then call startActivity(), passing the Intent object.
5. This is also a good time to add the About box that is required of all projects in this class. See the notes at the end of lecture module 2 for this.
6. Build and test your code on a Nexus 4 emulator with an API 16 image (instructions in Lab 0), or on your actual device. Make sure the buttons work. If they don't, you may have forgotten to register as an event listener or forgotten to enter your callback function in the XML layout (depending on which method you used for the handler).

Task 2: Add a second activity to your project

1. You can add a template activity, including an XML layout and the necessary entry into the Manifest, by right-clicking on your app directory from inside Android Studio. Do so, choosing Blank Activity, and call the activity SurveyActivity.
2. Modify the contents so it contains the widgets shown above, except that the TextView that says "Hello paul" above will initially just say "Hello". You'll be modifying the text of that widget programmatically. Make sure the input type for the age field is unsigned numeric, so that it launches a numeric keypad as opposed to an alphanumeric keypad.
3. Back in your main activity, modify your handler for the Take Survey button so that it grabs the text of the EditText field and converts it to a String using the String.valueOf() method. If the string is empty, display a Toast asking the user to please first enter a name. As an alternative you can initialize the button to a disabled state and enable it after the user enters a name, but that is a bit more challenging to accomplish. If a name has been entered, create an

Intent of type SurveyActivity.class, using "this" as the context. Pass the name string as a parameter to the intent using Intent.putExtra() and pass the intent to the startActivityForResult() method, which will launch the activity.

4. In your survey activity, modify the onCreate() method so that it retrieves the Intent and then retrieves the passed name parameter from the Intent. Grab a reference to the TextView object that provides a greeting (the widget you initialized to "Hello"), and set its text to "Hello " concatenated with the name that was passed via the Intent.
5. Add a handler for the Submit button. It should first check to see if the age field is empty. If it is, it should display a toast asking the user to please first enter their age. Again, you can accomplish this via button enables and disables if you prefer. If the age has been entered, parse it to an integer value, create a result Intent (use the no-argument constructor), and call putExtra() to attach the age value to the Intent. Then call setResult() with a value of Activity.RESULT_OK, and call finish() to end the activity and return to the "caller." If it is not clear to you why I put that in quotes, then you'd better come talk to me.
6. In your main activity, override the onActivityResult() method. In your implementation, start by checking the passed requestCode to make sure it matches the code you passed when you started the activity above. Also check to make sure the resultCode is equal to Activity.RESULT_OK. If both are the case, then call getIntExtra() on the passed data object. If the value is less than 40, change the content of the message field (initialized to "results will appear here") to "You're under 40, so you're trustworthy". Otherwise, change it to "You're NOT under 40, so you're NOT trustworthy".
7. Remove the settings menu from your survey activity, as it is not needed.
8. Build and test your code on either the Nexus 4 emulator you created in Lab 0, or on your actual device. Make sure the added survey activity works as intended.

Task 3: Handle Send Intents from Other Apps

1. Add statements to your AndroidManifest.xml that declare your app as able to handle SEND Intents with a data mimeType of "text/plain" in the DEFAULT category. There are notes on this in lecture module 2.
2. Modify the onCreate() method of your main activity so that it gets the Intent associated with the launch of the activity. If the intent is not null, get the associated action and type strings from it. If the action string is equal to Intent.ACTION_SEND and the type is "text/plain", set the text of the message widget (the one you initialized to "(results will appear here)") to the URL address extracted from the intent. See the Intent class for some standard keys.
3. Test that your app can respond to an Intent launched by another app (for example, the Share Page menu selection from a web browser). Make sure that what you display in your message widget is the URL address, not something else extracted from the intent.
4. Make sure an icon for your app shows up in the launcher. If it doesn't, you declared your SEND intent improperly. One of your activities needs two intent filters (for action types MAIN and SEND), and only one action is allowed in each intent filter. In addition, intent filters need to show up inside the tags for the Activity you want to associate with the intent.

Task 4: Instrument onCreate events with System notifications

1. Add a new class to your project called TracerActivity. Perhaps the easiest way to do this is to right-click on your project's java folder and choose "New→Java Class". Derive the class from whatever subclass of Activity the wizard generated for your main activity.
2. Give the class a notify method, which can be called to send a system notification to the user's device. The notification method will have a title with the passed string, and a content equal to the originating class name. It will also identify the time of the notification. Because this lab is already long enough, I am giving you the following code for this. When used judiciously, it can be useful for letting your user know about critical events in your application. We use it here to investigate the ordering of lifecycle events:

```
private void notify(String msg) {
  String strClass = this.getClass().getName() ;
  String[] strings = strClass.split("\\.") ;
  Notification.Builder notibuild = new Notification.Builder(this) ;
     notibuild.setContentTitle(msg) ;
     notibuild.setAutoCancel(true).setSmallIcon(R.mipmap.ic_launcher) ;
     notibuild.setContentText(strings[strings.length-1]) ;
  Notification noti = notibuild.build() ;
  NotificationManager notificationManager = (NotificationManager)
                      getSystemService(NOTIFICATION_SERVICE) ;
  notificationManager.notify((int) System.currentTimeMillis(), noti) ;
}
```

3. Now override the onCreate() method. The first order of business in your override should be to call the superclass version so that all the normal business of Activity creation gets done. Then test the value of the passed Bundle object. If it is null, make a call to a notify() method passing the string "onCreate NO state". Otherwise, make a call to notify() with the string "onCreate WITH state" and iterate through the key set attached to the bundle with additional calls identifying each key that is found in the set. Iterating through the keys can be accomplished with the following code:

```
Set<String> keys = savedInstanceState.keySet() ;
Iterator iter = keys.iterator();
while (iter.hasNext())
   notify("key:" + (String)iter.next()) ;
```

4. Add similar overrides to the onPause(), onRestart(), onStart(), onSaveInstanceState(), onStop(), and onActivityResult() methods. In these cases all you need to do is call the superclass version and provide a notification as to the method you are in.
5. Now change your main activity class so that it is derived from TracerActivity. Build and run your code. If your About box disappears, go back and reread step 1. If you forgot the name of the subclass that your MainActivity was originally using, you can probably still find it in your imports. Or look back at your Lab0 code.
6. Answer the questions on the last page of this lab and turn them in along with your apk file. If you zip them together, give your file a name in the form pschimpfLab1.zip. If you don't zip your files together, make sure they individual use the same naming standard, e.g., pschimpfLab1.apk and pschimpfLab1.doc (or .txt or .pdf).

Questions:

1. Run your app on your API 16 emulation. Examine the notifications by pulling down the notification bar. Which class are they from and do they indicate existing application state or not?
2. Dismiss the current notifications at this point so you don't get confused. Press Ctrl+F11 to rotate the Android and examine the system notifications again. In what order were the lifecycle methods that you instrumented called?
3. Do you notice anything odd about the notifications you posted from onCreate? What keys indicating state information are stored in the Bundle passed to onCreate? It's not anything you stored in the Bundle, so can you guess what that content represents?
4. Rotate back. Clear the notifications. Enter a name and press the Take Survey button. Enter an age and press Submit, taking you back to your main activity. Was your activity re-created this time? How do you know?
5. Clear the notifications. Press the Goto Website button and share the page with your App. What does the content of your main screen, along with the system notifications, tell you about this instance of your app?
6. Press the Android back button. Where does that take you? Press it again. Where are you now? Press it again. Where are you now? Does all of that make sense to you?
7. Repeat the experiment in step 5, making sure that the First Name field contains an entry before going to the website. This time, after sharing the page back to your app, press the Android recent apps button. Does the list of recent apps match what you would expect given your previous results from steps 5 and 6? Press the Android home button, and then press the recent apps button again? Now does it match your expectations? Note carefully the app names in addition to the app pictures.