LabW02 – Handling Interactions

Objectives:

1. Understand the basics of Model-View-Controller
2. Use click event listener for ListView
3. Use UI element dialog
4. Use Activity navigation
5. Experience Cross-Platform app development [Optional]

Tasks:

1. Handle long click event for an item in ListView
2. Pop up a dialog
3. Handle item clicking in ListView with Intent and another Activity
4. Develop Cross-Platform app with Flutter [Optional]

Assignment 1

- To develop a basic ToDoList app
- Worth 5 marks
- Due for submission by 7pm Monday 14/9 and demo in Week 04 lab, with project files zipped as 1 zip file and submitted in Canvas

Good mobile apps provide useful functionality and an easy to use interface. The user interface is made up of various Graphical User Interface (GUI) components and typically waits for user interaction.

In this tutorial, we will learn about various GUI components, their associated events, and how to handle events on those components. You should finish LabW01 before proceeding with the following tasks:

Task 1: Handling item long clicking in ListView

Currently nothing happens when an item is clicked in the ListView. We will first implement "deleting an item" by long clicking it.

1. Add the following method into the MainActivity.

   It sets two listeners for LongClick and Click events.

```
private void setupListViewListener() {
    listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
        public boolean onItemLongClick(AdapterView<?> parent, View view, final int
position, long rowId)
        {
            Log.i("MainActivity", "Long Clicked item " + position);
            return true;
        }
    });

    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
            String updateItem = (String) itemsAdapter.getItem(position);
            Log.i("MainActivity", "Clicked item " + position + ": " + updateItem);
        }
    });
}
```

2. Call this method at the end of the onCreate() method

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

    // Setup listView listeners
    setupListViewListener();
}
```

3. Run it. Click and long click a ToDoItem in the list view.

4. Now update the listener for onItemLongClickListener.
   Add two lines of code to remove an item from the ArrayList and notify the ListView adapter to update the list. Run it.

```
        Log.i("MainActivity", "Long Clicked item " + position);
        items.remove(position); // Remove item from the ArrayList
        itemsAdapter.notifyDataSetChanged(); // Notify listView adapter to update list
        return true;
```

Run your code, and long click a ToDoItem, you should able to see the following line printed on the Android Studio Logcat:

```
comp5216.sydney.edu.au.todolist I/MainActivity: Long Clicked item 1
```

Task 2: Popping up a dialog

1. Add a dialog to let user confirm the delete operation. Replace the existing OnItemLongClick method.

```
listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    public boolean onItemLongClick(AdapterView<?> parent, View view, final int
position, long rowId)
    {
        Log.i("MainActivity", "Long Clicked item " + position);
        AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setTitle(R.string.dialog_delete_title)
                .setMessage(R.string.dialog_delete_msg)
                .setPositiveButton(R.string.delete, new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialogInterface, int i) {
                        items.remove(position); // Remove item from the ArrayList
                        itemsAdapter.notifyDataSetChanged(); // Notify listView adapter
to update the list
                    }
                })
                .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialogInterface, int i) {
                        // User cancelled the dialog
                        // Nothing happens
                    }
                });

        builder.create().show();
        return true;
    }
});
```

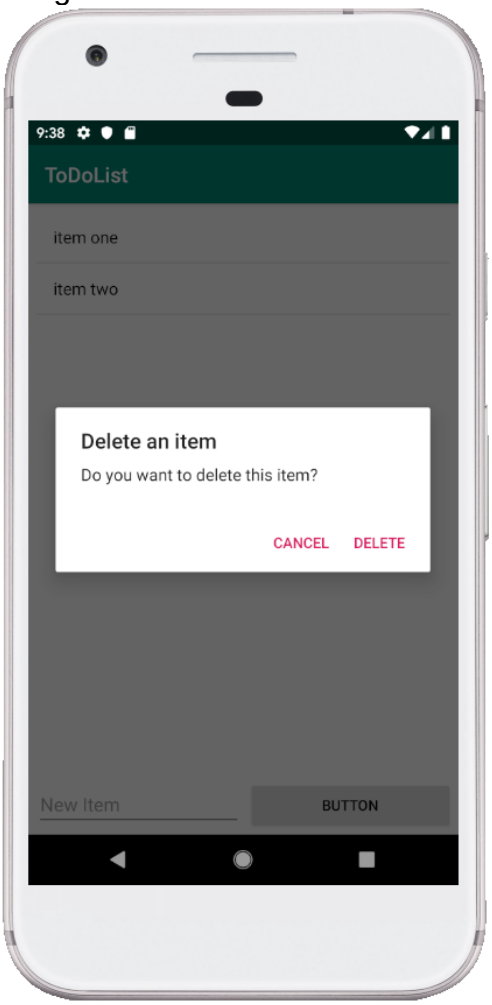
You will also need to add the following String resources into "**res/values/strings.xml**"

```
<string name="dialog_delete_title">Delete an item</string>
<string name="dialog_delete_msg">Do you want to delete this item?</string>
<string name="delete">Delete</string>
<string name="cancel">Cancel</string>
<string name="edit">Edit</string>
```

Run it and long click an item.

2. In order to further customise a dialog, such as adding a third button, or using your own layout, please refer to the following tutorial: https://developer.android.com/guide/topics/ui/dialogs

   If you do the above steps right, you should able to see the following screenshot once you long click a ToDoItem:



Task 3: Handling item clicking in ListView with Intent and another Activity

When clicking an item (NOT a long click), we should open another Activity to edit the item.

1. Copy the EditToDoItemActivity.java to the same "**src**" folder as the MainActivity.
   This Activity receives the data from the MainActivity when an item is clicked, and send back the updated content to the MainActivity. Read the inline comments for details.

2. Copy **activity_edit_item.xml** to the "**res/layout**" folder. This is the layout for EditToDoItemActivity

3. Update the onItemClick method in MainActivity to open the EditToDoItemActivity

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    String updateItem = (String) itemsAdapter.getItem(position);
    Log.i("MainActivity", "Clicked item " + position + ": " + updateItem);

    Intent intent = new Intent(MainActivity.this, EditToDoItemActivity.class);
    if (intent != null) {
        // put "extras" into the bundle for access in the edit activity
        intent.putExtra("item", updateItem);
        intent.putExtra("position", position);
        // brings up the second activity
        startActivityForResult(intent, EDIT_ITEM_REQUEST_CODE);
        itemsAdapter.notifyDataSetChanged();
    }
}
```

When an item is clicked, it creates an Intent to start another activity and wait for its result. You need to define a variable to remember the request code in the MainActivity.

```
public final int EDIT_ITEM_REQUEST_CODE = 647;
```

4. When the EditToDoItemActivity finish, MainActivity will receive the result by checking the request code first, and then uses its result.
   Add the following method into MainActivity:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == EDIT_ITEM_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Extract name value from result extras
            String editedItem = data.getExtras().getString("item");
            int position = data.getIntExtra("position", -1);
            items.set(position, editedItem);
            Log.i("Updated Item in list:", editedItem + ",position:"
                    + position);

            Toast.makeText(this, "updated:" + editedItem, Toast.LENGTH_SHORT).show();
            itemsAdapter.notifyDataSetChanged();
        }
    }
}
```

To learn more about using intents to create flows, please read:
https://github.com/codepath/android_guides/wiki/Using-Intents-to-Create-Flows

You may also notice the use of Toast. It is a simple pop-up notification method in Android. Learn more at:

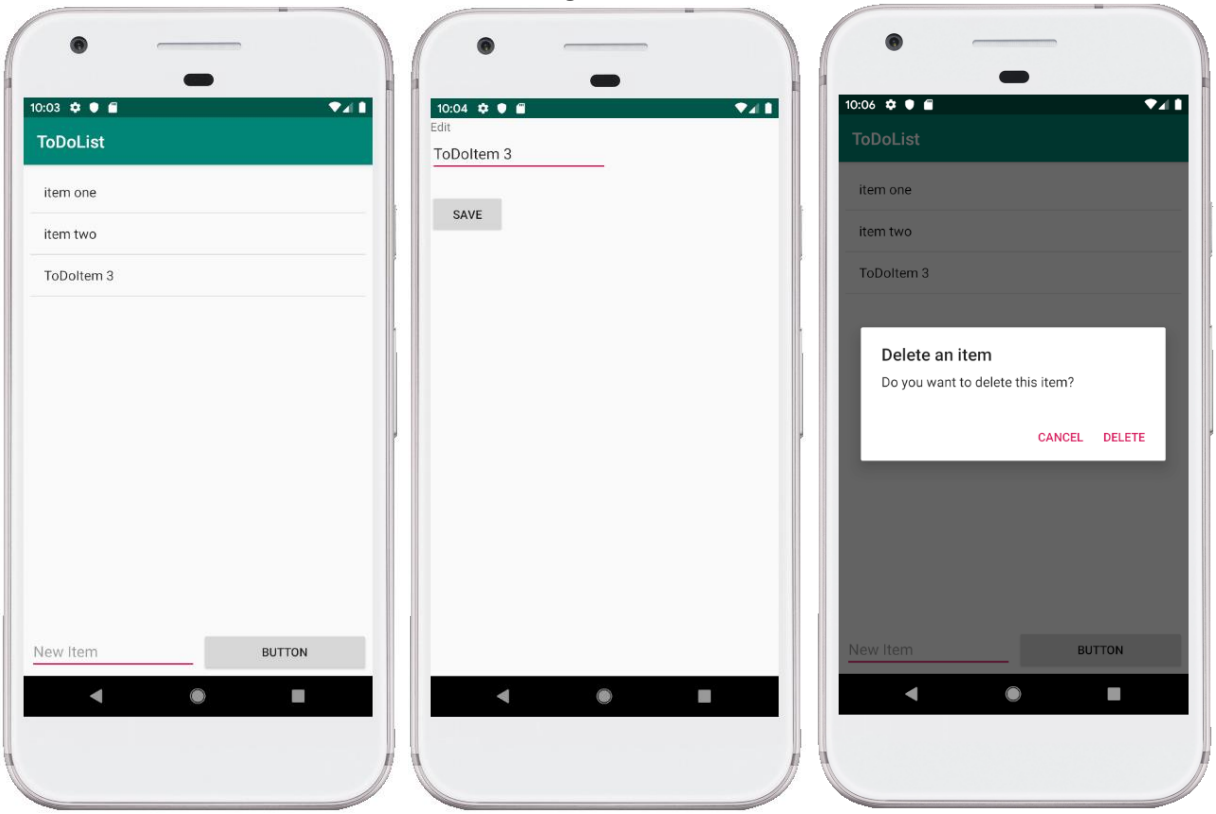
https://developer.android.com/guide/topics/ui/notifiers/toasts

5. Lastly, the application manifest must include the new EditToDoItemActivity, otherwise this Activity cannot be used.

   Open **AndroidManifest.xml** and add the following code inside the <Application> tag.

```
<activity
    android:name=".EditToDoItemActivity"
    android:label="@string/app_name" >
</activity>
```

6. Run it. Test the above steps. If all the steps are correct, your app now are able to handle the click and long click events:



Task 4: Using Flutter [Optional]

Flutter is a tool introduced by Google in 2018 for cross-platform Android and iOS mobile apps development. It is considered as one of the most efficient tools to build platform independent apps quickly, with a single codebase for both Android and iOS. This means that you can use one programming language and one codebase to create two different apps (for Android and iOS).

Flutter is an open-source mobile application development framework, available for download from: https://flutter.dev/docs/get-started/install

To get started, select the operating system on which you are installing Flutter:

1. Flutter for Windows install: https://flutter.dev/docs/get-started/install/windows
2. Flutter for MacOS install: https://flutter.dev/docs/get-started/install/macos
3. Flutter for Linux install: https://flutter.dev/docs/get-started/install/linux

Please refer to the following URL for more details on Flutter: https://flutter.dev/