MOBILE DEVICES – ASSIGNMENT 2

Mobile Devices – Assignment 2 DevanSuperNotes

Submitted: November 07, 2014
By: Devan Shah 100428864
Submitted to: Michael Chang

Design Techniques

For assignment 2 I have constructed a notpad mobile application in Android using eclipse. Using the interface builder in eclipse for Android I was able to construct the UI design of the Android notes application. My notes application is based off of the default iOS notes application, with the exception of changes in the design to be portable on Android and also reduce meant of the features and the way it works, but the basic concept applies. Before going forward and constructing the design in eclipse I constructed a sample design sketch to see how the notes application would look on a phone. I also added more functionality then required by assignment 2 instructions and also support landscape orientation for most of the views. Also when the orientation is changed it will preserve the data and restore the orientation based of the preserved data. My note application consists of a simple design with two main transition aspects (views) to move from list of notes to view notes description or create new notes. My design made use of one activity and having the 2 fragments to switch between the views listed above. The list view of notes was implemented as a fragment who's main goal is to make use of a SimpleAdapter to populate the list view information in a appropriate manner (ordering by time) and handle onClick. The notes description was implemented as a fragment also, where the main goal was to display the title of the selected note and note content, this fragment was also uses to add new notes. My notes application also supports presentence storage to save the notes that are created so that they can be restored when application is terminated or stopped. This allows for notes to be saved over time until the application is manually uninstalled.

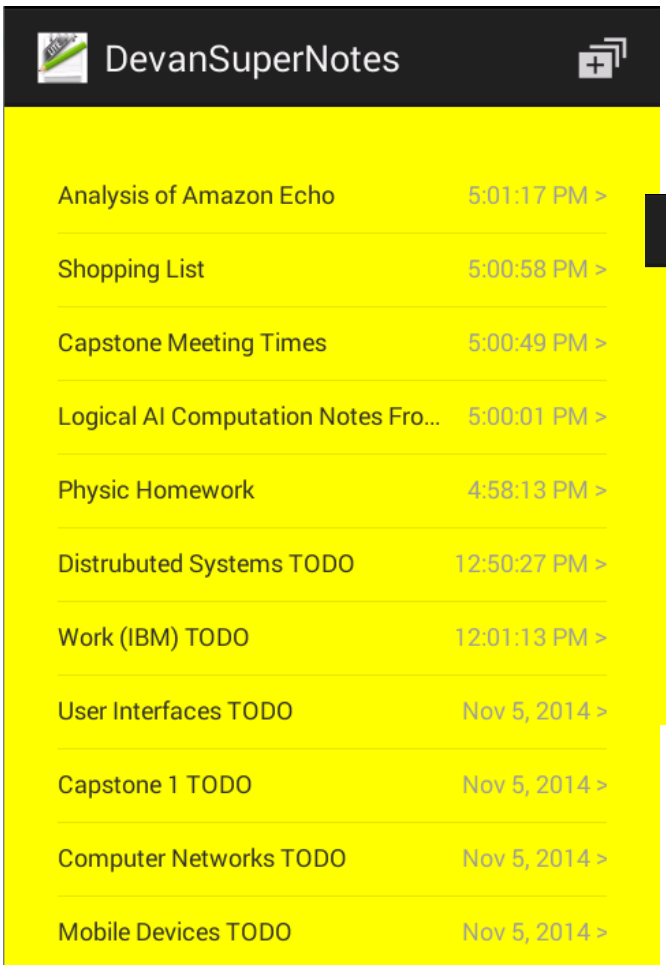

Figure 2 Android Note UI Design portrait view. Shows the list view with multiple notes and also the add button on the right top corner.

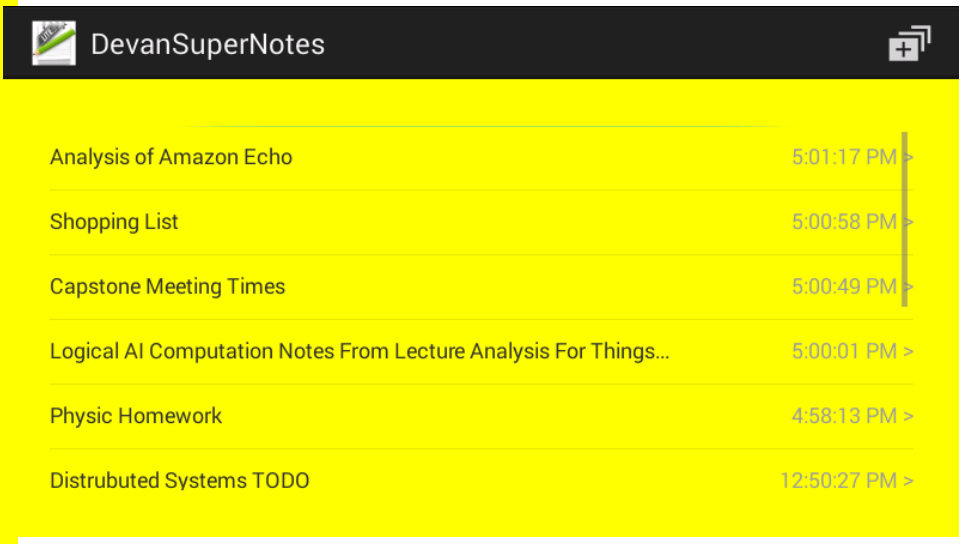

Figure 1 Android Note UI Design landscape view. Shows list view as scrollable in case all notes don't fit on screen.

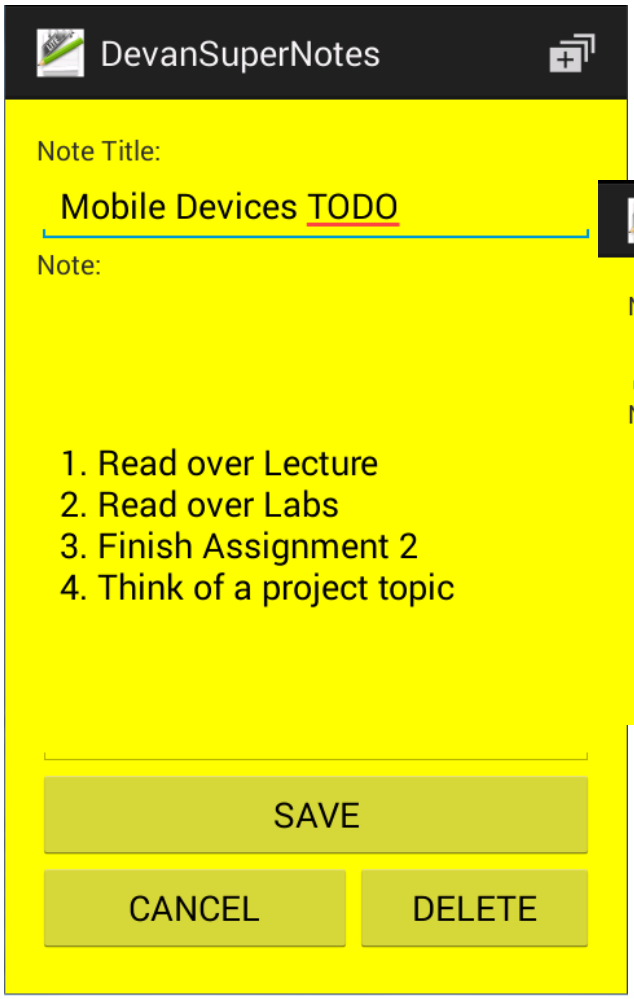

Figure 3 Android Note UI Design portrait view. Shows the note description when selected from the list view, this view also supports adding new messages.

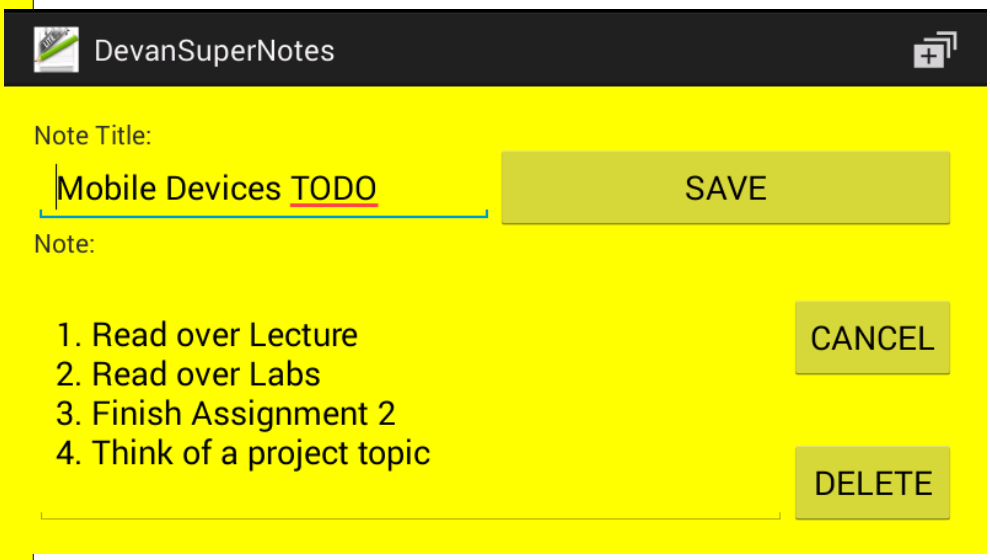

Figure 4 Android Note UI Design landscape view. Shows the note description when selected from the list view, this view also supports adding new messages.

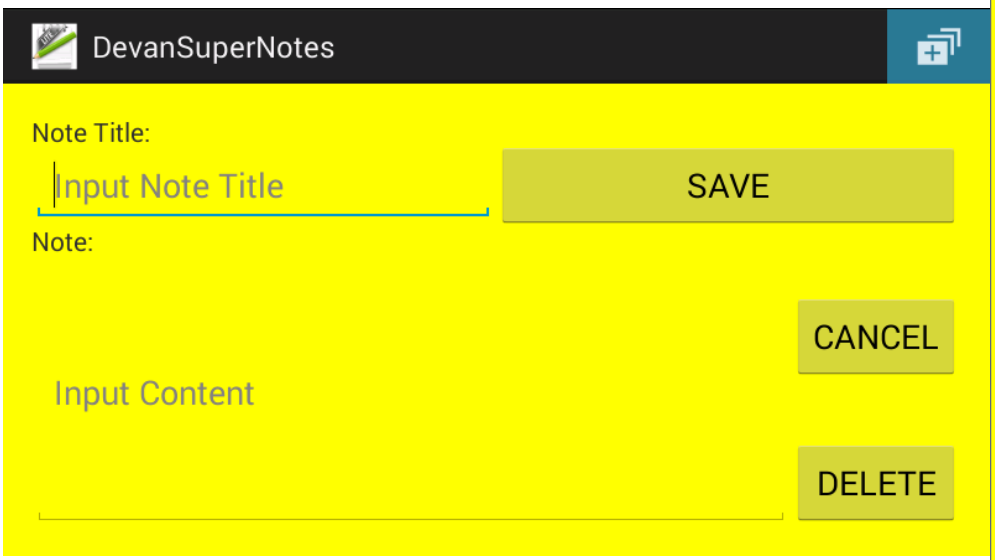

Figure 5 Android Note UI Design landscape view. Shows add a new note UI when the add button is clicked from the action bar menu.

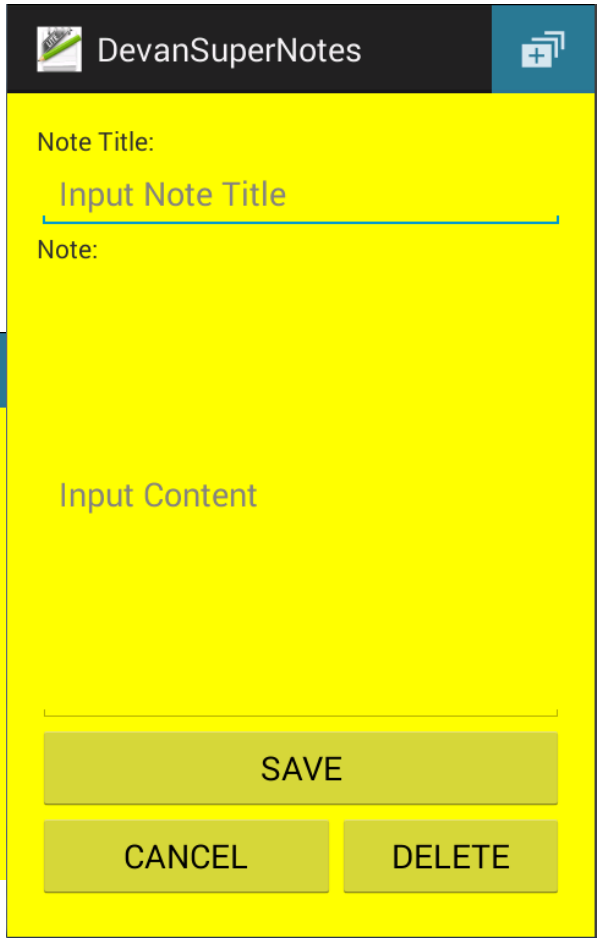

Figure 6 Android Note UI Design portrait view. Shows add a new note UI when the add button is clicked from the action menu bar.

Implementation Techniques

The Android note's application was implemented in eclipse using the android application development suite and the use of java. Java played a crucial role in implementing the notes application and having all the operations performed with the use of build in java libraries and default Android objects. I have implemented multiple functionalities in the notes application which allows it to be robust and perform the most basic operations for saving and deleting notes. I have implemented couple of error checking mechanisms that are used to make sure errors are prevented down the road.

Furthermore on the side of coding techniques I utilized multiple resources on Android development to learn about most efficient ways to implement aspects of the notes application. Couple neat methods that I used to achieve a fully working notes application where the following, Android internal storage, utilized multiple different data structures for storing and displaying information, dialog boxes on Android, different properties of editboxes, list views, text views and fragments. The internal Android storage was used to store the Vector<Object> of notes into a file for restore when app crashes or on a new restart. I used multiple different data structure and combinations of data structures, Vector<Object> was used to store the notes that were added/deleted, <ArrayList<Map<String, String>> was used store the notes in a form that could be passed to the list view. Made use of Android dialog boxes to make sure that the user enters a title when creating new notes or editing notes. This was added to make sure that there are no blank lines showing up in the list view. Make use of multiple different xml properties for boxes to cap number of lines, amount of text displayed at once, show there is more text when box length is reached, and different types of input. Furthermore, I made use of fragments for navigation between list of notes, note description and create new note, the fragments are replacing one FrameLayout in the MainActivity (only use one activity). There were multiple different techniques used to achieve a working version of the notes application.

```
public void saveNotesInInternalStorage()
{
    // variable deceleration
    ObjectOutputStream notesRawDataOut ;

    try {
        // Construct the stream to write the vector of notes saved already.
        notesRawDataOut = new ObjectOutputStream(new FileOutputStream(
                notesRawDataFile.getAbsoluteFile()));
        notesRawDataOut.writeObject ( notes ) ; // Write the object
        notesRawDataOut.flush() ; // flush the stream to make sure everything is written.
        notesRawDataOut.close() ; // Close the stream
    } catch ( IOException e ) { e.printStackTrace() ; }
```

Figure 8 Shows the saveNoteInInternalStorage function, this function was used to save the notes to android device's internal storage.

```
public void restoreNotesFromInternalStorage()
{
    // Create the File handle for the file that is checked if an restore is needed
    notesRawDataFile = new File ( getFilesDir(), "NotesRawData.ser" ) ;

    // Perform the restore only if the file exists.
    if ( notesRawDataFile.exists() )
    {
        // Variable deceleration
        ObjectInputStream notesRawDataRestore;

        try
        {
            // Open the stream to retrieve the saved notes from the file.
            notesRawDataRestore = new ObjectInputStream(
                    new FileInputStream(notesRawDataFile.getAbsoluteFile()));

            // Read the data in the file and store it in the notes vector.
            notes = ( Vector<Object> ) notesRawDataRestore.readObject() ;

            notesRawDataRestore.close() ; // Close the stream.
        }
```

Figure 7Shows the restoreNotesFromInternalStorage function that was used to restore the notes data structure from the file stored on intern storage.

Figure 8 and Figure 9 shows snippet of functions that are used to store and read data from Android internal storage for the application. Each application has a dictated internal storage, where application dependent files are stored. I am making use of this functionality to store the notes when new notes are added/deleted. This makes sure that the notes for users are always up to date on opening application again after closing.

```
// Date Structure Deceleration
public Vector<Object> notes;

// Object Deceleration
public NotesDescription myNotesDescription;
public NotesList myNotesList;
public File notesRawDataFile ;
```

Figure 10 Shows the data structure Vector<Object> that holds the notes object that users add.

```
// Date Structure Deceleration
ArrayList<Map<String, String>> notesExtracted;
Vector<Object> notes;

// Stores the main activity
Activity myNotesStartActivity;
```

Figure 9 Shows the data structure ArrayList<Map<String, String>> that is used to store the notes in a form that can be printed to the list view in a proper manner.

Figure 10 and Figure 11 the data structure that I used in my notes application, these are the main data structures that are used to store a form of the notes object. These are used to store notes that users add/delete and also used to display the notes in a list view with the use of a SimpleAdapter that would use a layout for each of the rows to display the note title and the time created/updated. Figure 11 shows a snippet of how this setup functions to create the list view.

```
notesExtracted = new ArrayList<Map<String, String>>();
buildNotesArray();
SimpleAdapter notesAdapter = new SimpleAdapter ( myNotesStartActivity,
                                                 notesExtracted,
                                                 R.layout.listviewrow,
                                                 new String[] { "Title", "Date" },
                                                 new int[] { R.id.TITLE_CELL, R.id.DATE_CELL }
                                               );
ListView listView = (ListView) view.findViewById(R.id.listView1);
listView.setAdapter(notesAdapter);
listView.setOnItemClickListener(noteSelected);
```

Figure 10 Shows how the data structure is used to display the notes in the list view. The SimpleAdapter uses the list view layout, header of title and date in the form of an array of strings and array of int with the id of the text views the data will be put in.

```
// Loop through the note vector and build a HashMap and add it to the ArrayList
for ( int i = 0; i < notes.size(); i++ )
{
    // Create a new HashMap
    noteMap = new HashMap<String, String>();

    // Grab the notes entry
    Notes noteEntry = ( ( Notes ) notes.elementAt (i) ) ;

    // Put the Title and Date from the notes object into the HashMap
    noteMap.put ( "Title", noteEntry.getNoteTitle() );
    noteMap.put ( "Date", noteEntry.getNoteCreationDate() );

    // Put the HashMap into the ArrayList
    notesExtracted.add(noteMap);
}
```

Figure 11 Shows how the ArrayList<Map<String, String>> is created from the Vector<Object>.

Figure 12 shows the loop that I used to construct the ArrayList<Map<String, String>> from the Vector<Object>. The loop runs through the notes and extracts the Title and the Date from the notes object and puts it into the HashMap using keys Title and Date. (Theses keys are used by the SimpleAdapter to construct the list view.

Figure 11 shows the snippet of code that I implemented to create the dialog box when the user tries to create note with an empty title. This is to avoid future failures.

```
new AlertDialog.Builder(myNotesStartA
.setTitle("Empty Note Title")
.setMessage("Please enter a Note Title before Saving. Thanks")
.setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
        dialog.cancel();
    }
 })
.setIcon(android.R.drawable.ic_dialog_alert)
.show();
```

Figure 12 Shows how the dialog is constructed when the Title field of a note is left blank.

I implemented a Notes object that is used to store all the information for one single note the user constructs. The Notes object stores the title, content and created data of the note, I have also implemented multiple functions that are used to retrieve specific aspects of a note. My Notes class also implements Serializable and Comparable<Notes>, the Serializable is used so that the Notes object can be stored in the file as serialized object and it also implements Comparable<Notes> so that I can compare the data of Notes object to determine the order in which to display the notes in the list view.

```
public class Notes implements Serializable, Comparable<Notes>
{

    * Default serialization constant for this object.
   private static final long serialVersionUID = 1L ;
```

Figure 13 Shows the implementation of my Notes object.

```
public Notes ( String noteTitle, String noteDescription) {
    this.noteTitle = noteTitle;
    this.noteDescription = noteDescription;
    this.noteCreationDate = new Date() ;
}
```

Figure 14 Shows the data that is stored for each of the notes the user creates.

Some of the function that I have implemented in the Notes class include: getNoteTitle(), getNoteDescription(),getNoteCreationDate(), and compareTo(Notes another).

Following is some sample images of features that were implemented aside from design implementations mentioned above.

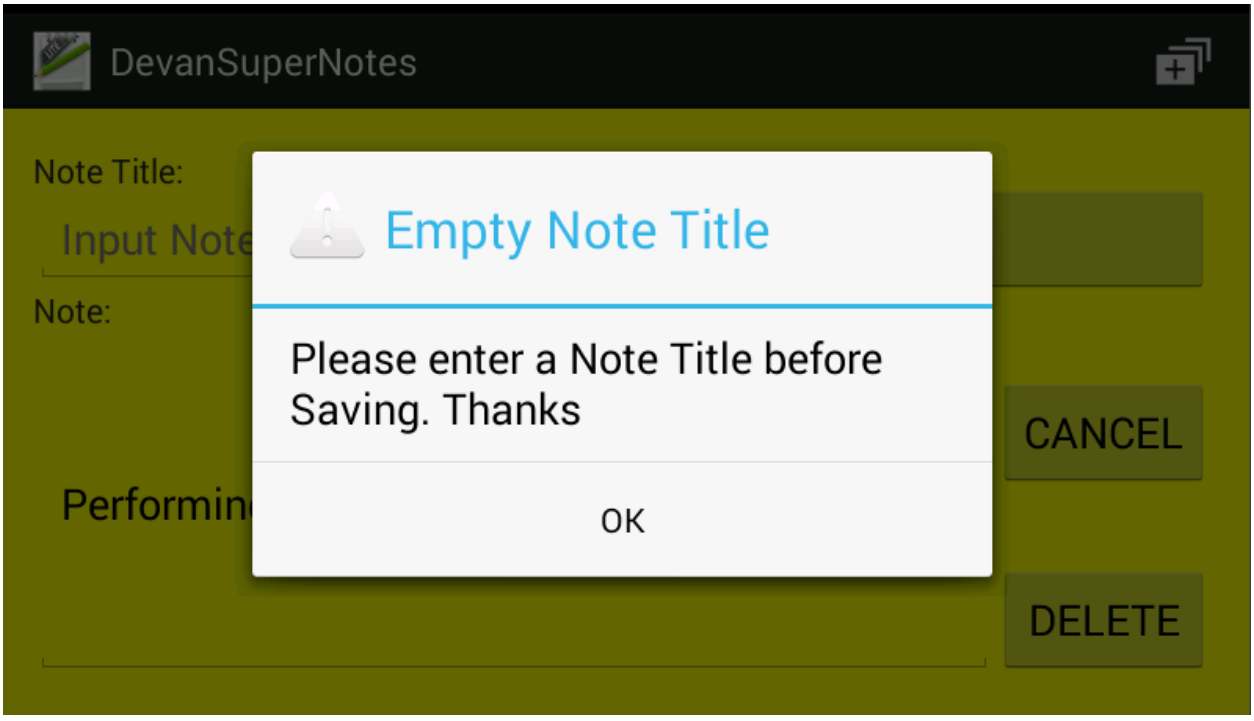

Figure 15 Shows the dialog window that opens up when the user does not input a tile for the note.

Furthermore, the functionality is pretty self-explanatory when the note application is being used. The add button at the top can be used to add new notes, the save button on the add new note/edit note will save the notes to vector and file, the cancel button on the add new note/edit note will return user back to the list view without performing any operations and the delete button on the add new note/edit note will delete the note that was selected or discard the values that are entered for new notes.