**Student Written Assessment (Assignment)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Business Unit/Work Group** | IT Studies | | |
| **Qualification Code** | ICT50715 | **Qualification Title** | Diploma of Software Development |
| **Unit Code/s** | ICTPRG527 | **Unit Title/s** | Apply intermediate object-oriented language skills |
| **Assessment Task Title** | GroupSMS Assignment | | |
| **Student Name** | Submit your solution via your LEARN account | **Student SIS ID** | Submit your solution via your LEARN account |
| **Assessor Name** | You have been added to a LEARN group which defines your assessor. This is normally your Course Registration Number (CRN) lecturer. | **Date** | 2018 Semester 2 |

|  |  |
| --- | --- |
| **Student Guide for Written Assessment (Assignment)** | |
| **Overview of Assessment** | This assessment will require you to complete a partially implemented version of an app that will send SMS to a collection of phone numbers. The app is called GroupSMS.  In this assessment you will cover the following topics:   * User Interface Creation * Intents * Using Lists * Data persistence in a database   The assessment is broken up and assessed in 3 parts and you will submit each part gradually as you develop your skills through the course. The final part will be a complete working version of the GroupSMS app |
| **Task/s to be assessed** | This is a practical task and will require you to code, test and document in the Java language in an Android context. The following pages will define the specific tasks you need to complete. |
| **Time allowed** | You have the whole course duration to complete this assignment, however each part has its own submission date.  The submission dates for each part are provided in the LEARN course and summarized in the Assessment Submissions topic in LEARN. |
| **Location** | You can complete this assessment during your practical sessions and at home. |
| **Decision making rules** | To receive a satisfactory outcome for this assessment you must complete all parts correctly. |
| **Assessment conditions** | You must follow the provided ITWorks Android coding standards in completing this assessment. |
| **Resources required** | To complete this assignment, you will need to use Android Studio (latest version) and have been provided with an Android Studio project containing portions of the code. Android Studio and Windows based machines are provided in your practical classes. You can use a Mac if you prefer but these are not provided in the classrooms. |
| **Results/Re-assessment** | You will be provided feedback for each part of the assessment and be given the opportunity to resubmit with any required corrections only once. |

# GroupSMS Assignment

## OBVERALL SCENARIO

You have been employed by the ITWorks organisation as a junior Android programmer. Your job is to complete some components for the partially developed GroupSMS app.

Others have already worked on this app and some code has been developed. Your role will be to complete some of the components and make some enhancements as guided by the following pages.

The overall purpose of the app is to allow the use of an Android device to send an SMS to a group, that is, multiple phone numbers.

You must follow all ITWorks organisational standards.

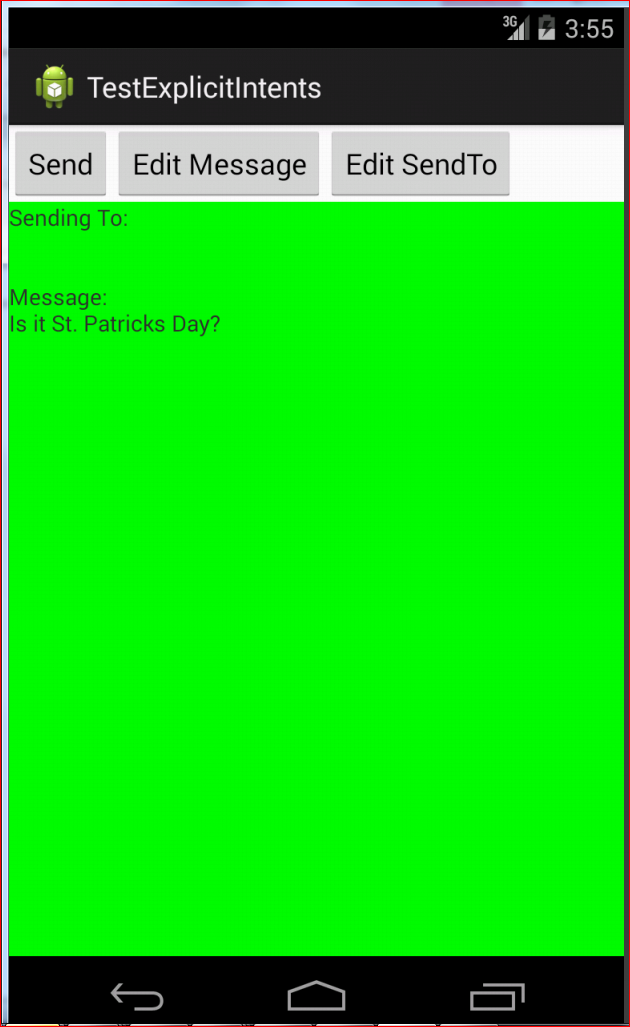
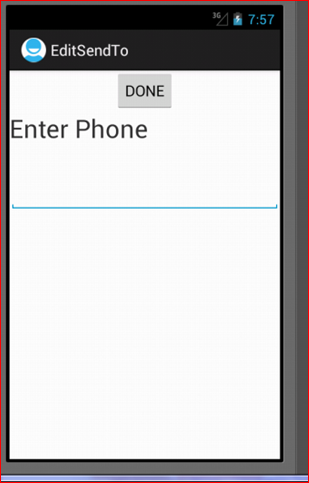
# Part 1 – Basic GUI

## Scenario

A previous programmer produced a proof of concept for a basic first version of GroupSMS, in a project called GroupSMSVersion1, which only allows sending an SMS to one phone number. However, they only completed the Send Message use case. Use GroupSMSVersion1 as a starting point to write a new version of the GroupSMS app to meet the following requirements. Call this project GroupsSMSVersion2

* **Program Requirements**

There are two activities and the output should look as per the following mock-up. The “Edit Send To” button starts another Activity called EditSendTo that will allow the data entry of the phone number. We will keep the EditSendTo interface simple for now by just working with one phone number (see below) but in a future more advanced GUI release we will be adjusting EditSendTo to allow multiple phone numbers to be entered. The app must be developed for Android Version 5.1.1 (Lollipop).



“Edit Send To” starts this Activity.

The phone number entered comes back and is shown here

## Assessment Requirements

* **Coding**

Code the solution that meets the above requirements and ensure that

* You use LinearLayouts (within an overarching ConstraintLayout) in EditSendTo
* All views have appropriate ids and all strings are defined as resources as per the ITWorks required standards.
* Set the inputType so that only legal characters for a phone numbers are allowed as input.
* Use the Intent object to communicate the data between Activities
* **Best Practices**:  
  Follow the provided ITWorks organisational guidelines for developing Android code
* **Testing**  
  To demonstrate that you have adequately reviewed the program requirements and produced the required functionality you must document some testing by submitting a word document containing screen shots of the running of your complete solution. Call the word document Part1\_Test\_Results\_YourName.docx. The screen shots need to validate that the following aspects of the above requirements have been met:
  + Selecting Edit Message opens the EditMessage Activity (which was provided) and allows the message to be edited and displays the new message on return to the starting GroupSMS Activity.
  + Selecting Edit Phone allows the phone to be edited and displays the new phone on return to the starting GroupSMS Activity along with the correct message (unchanged).
  + Selecting Send displays a Toast message indicating the message has been successfully sent and includes the required code to actually send the message.
* **Documentation**:  
  Create the Javadoc documentation for all methods in the classes developed. You must have the Javadoc folder in a folder called Javadoc at the first level of the project which contains the html files produced by the Javadoc tool.
* **Submission summary:**
* Zip file of your Android Studio project with all classes completed. You may delete any components of the dist (distribution) folders to reduce the size of the zip file.
* Word document called Part1\_Test\_Results\_YourName.docx following the ITWorks style guidelines for simple reports and containing screen shots of the running of the test classes.

# Part 2 – Internal Storage of Collections of Data

## Scenario

The same previous programmer developed a component of the Model classes, the classes that store the data as the app is running. They provided an interface called SMSDataModelnterface and an implementation of that interface called SMSDataModelArray. However, ITWorks management have decided that a more efficient and modern version of the SMSDataModelnterface is required, one that uses a Java List, and fixes the bugs in the array version. You need to:

* **Testing**  
  Write a test class called SMSDataModelTest to test the methods in SMSDataModelArray. This will ensure you understand the purpose of the SMSDataModelInterface methods. Ensure that you declare relevant variables to be of type SMSDataModelnterface as you will be reusing the test class to test a different implementation. What each method is supposed to do is documented in SMSDataModelInterface.
* **Coding**  
  Write a new implementation of SMSDataModelInterface called **SMSDataModelList** that uses a List rather than an array to store the phone numbers and ensure you meet the following requirements:  
  + You write the code so that you can easily change from an ArrayList to a LinkedList and that the List can only store String objects.
  + Ensure that your solution makes use of all relevant List methods and the return results from those methods.
  + You introduce a new instance variable to record the maximum number of phone numbers that can be stored as the current version of the code relies on the length of the array to check this.
  + You remove any unnecessary instance variables, such as numPhoneNumbers, as the List size method provides this information.
  + Correct the handling of the model full case in the addPhoneNumber method. An Exception approach to handling the case when code tries to add more numbers than is allowed into the Model is meant to be used.
  + The model will never be full, unless the constructor which allows a maximum to be defined is used, this was never properly implemented in the array version.
  + Fix the bug in the method updatePhoneNumber in the provided Array implementation – it should not allow the update to cause there to be a duplicate phone number in the Model.
  + Complete the sortNumbers method which was never provided in the array version
* **Best Practices**  
  Ensure that you are following the ITWorks Java and Android coding standards.
* **Documentation**  
  Internal: Ensure the Javadoc comments for any new or changed methods has been updated. You must have a Javadoc folder which contains the html files produced by the Javadoc tool.  
    
  External: Create a UML class diagram to show the relationships between the model implementations (SMSDataModelArray and **SMSDataModelList)** and their relationship to the interface class (SMSDataModelnterface).
* **Submission summary:**
* Zip file of your Netbeans (or Android Studio) project with all classes completed. You may delete any components of the dist (distribution) folders except the javadoc folder to reduce the size of the zip file.
* Word document called Part2\_Results\_YourName.docx following the ITWorks style guidelines for simple reports containing screen shots of the running of the test class and the UML diagram.

# Part 3 – Advanced GUI – Using ListView

## Scenario

The client asked for EditSendTo to be upgraded to allow for multiple phone numbers to be specified.

The upgraded version of EditSendTo was worked on by a previous programmer but was not finished. You have now been asked by your ITWorks supervisor to complete the missing components and ensure that the functionality is as per the user request.

Below, under the heading, “Design Documentation and work from Previous Programmer” you will find information on what the user wanted and what the previous programmer got done.

## Assessment Requirements

You need to:

* **Coding**  
  Get the provided code working. Finish off the missing functionality – the “Add New” and the “Delete” do not work
* **Testing/Debugging**Ensure that the Add New button is adding to the model and that the ListView Adapter is used appropriately to update the view.  
    
  Ensure that the Delete button properly updates the “Update Entry X” button label and fill in the data in the EditText. The requirements are for the label and data to remain referring to the same data and its correct position in the list when a deletion occurs except when the item showing is the one deleted. Examples have been provided by the client in the “Design Documentation and work from Previous Programmer” section.  
  This aspect will be a focus and the submission will require you demonstrating stepping through the code logic for the deletions.
* **Best Practices**  
  Follow the provided ITWorks organisational guidelines for developing Android code.   
  **Ensure the new SMSDataModelList (that you developed) is used in this version.**

**Documentation**  
Create a test document with screen shots showing .

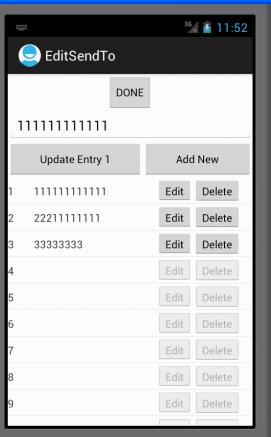
* + EditSendTo
    - Add New:
      * Successful usage of the “Add New” button
    - Delete:
      * Successful usage of the “Delete” button with at least 4 items in the list and showing the third item as selected then deleting the first item.
      * Successful usage of the “Delete” button with at least 4 items in the list and showing the third item as selected then deleting the last item.
      * Successful usage of the “Delete” button with at least 4 items in the list and showing the third item as selected then deleting the third item.
      * Screen shots of you stepping/tracing through the delete logic and watching the variables change. Ensure you capture the watch window in Android Studio in your screen shot.
  + GroupSMS:  
    Screen shot of GroupSMS Activity on return from EditSendTo correctly showing all phone numbers. Follow the provided ITWorks organisational guidelines for developing

**Submission summary:**

* Zip file of your Android Studio project with all classes completed. You may delete any components of the dist (distribution) folders to reduce the size of the zip file.
* Word document called Part3\_Results\_YourName.docx following the ITWorks style guidelines for simple reports and containing
  + screen shots of the running of the completed GroupSMS app as per the documentation section above.
  + The code for the “Add New” and “Delete” event handlers.

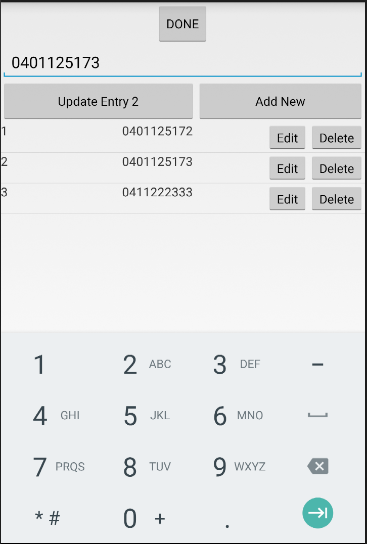
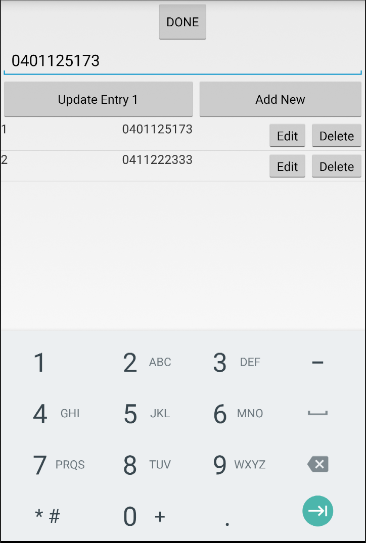
**Design Documentation and work from Previous Programmer**

This new version uses the new SMSDataModelList class to hold all information about a message and a new version of EditSendTo to allow multiple phone numbers to be entered. The following is a screen shot of what EditSendTo now looks like:

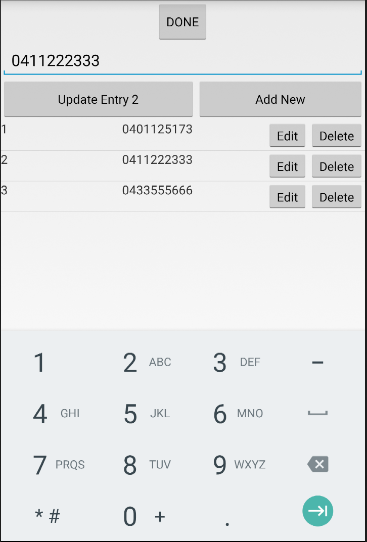
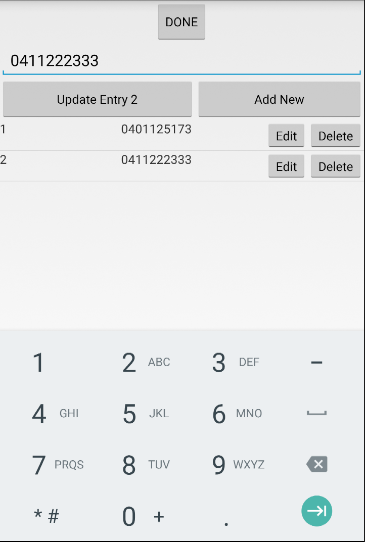


**Clients Expectation - EditSendTo interface.**

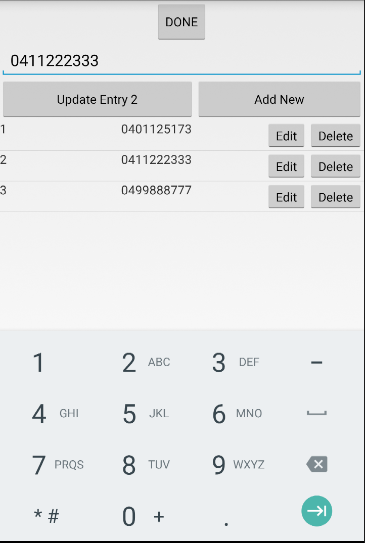
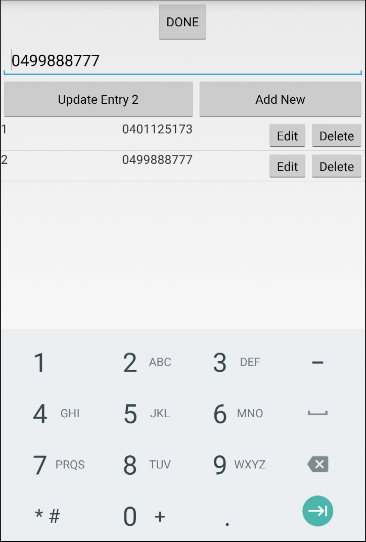
The following was provided by the client on how they expect the interface to change when items are deleted.

Item 1 deleted

Item 3 deleted

Item 2 deleted

**Previous Programmer Code.**

Some important aspects:

* An implementation of SMSDataModelInterface has been used to package all information about the message, including the message itself and the phone numbers to use. This has been placed in a package called models.
* All Activity classes have been placed in a package called activities.
* All event handling is done in named inner classes.

The following pages will discuss some important aspects of the code. It will not show all the code. The project in its existing state is available (download from Learn).

**GroupSMS.java (not all code is shown here, see download of files from Learn)**

package itstudies.tafesa.groupSMS.views;

import itstudies.tafesa.groupSMS.R;...

public class GroupSMS extends Activity {

Creates an SMSDataModel capable of storing up to 20 phone numbers. With a blank initial message.

Just for testing purposes we put in 2 dummy numbers. This would be removed in final version.

public static final String CLASS\_TAG = "GroupSMS";

public static final int NEW\_MESSAGE\_REQUEST = 1;

public static final int NEW\_SENDTO\_REQUEST = 2;

private SMSDataModelInterface messageData;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_group\_sms);

// Getting to the views defined in the XML files.

TextView textView1 = (TextView) findViewById(R.id.textView1);

textView1.setBackgroundColor(Color.GREEN);

textView1.setMovementMethod(new ScrollingMovementMethod());

messageData = new SMSDataModelList("", 20);

messageData.addPhoneNumber("00000000");

messageData.addPhoneNumber("11111111");

setSummary();

// Responding to an event - the onClick for the Edit Message Button using

// a named inner class

Button btnEditMessage;

btnEditMessage = (Button) this.findViewById(R.id.butEditMessage);

HandleButtonEditMessageOnClick buttonEditMessageOnClick;

buttonEditMessageOnClick = new HandleButtonEditMessageOnClick();

btnEditMessage.setOnClickListener(buttonEditMessageOnClick);

// The onClick for the Edit Send To Button using a named inner class but no

// variable

Button btnEditSendTo;

btnEditSendTo = (Button) this.findViewById(R.id.btnEditSendTo);

btnEditSendTo.setOnClickListener(new HandleButtonEditSendToOnClick());

// Send Button onClick Action starts the implicit intent to

// send an SMS.

Button btnSend;

btnSend = (Button) this.findViewById(R.id.btnSend);

btnSend.setOnClickListener(new HandleButtonSendOnClick());

}

public class HandleButtonSendOnClick implements OnClickListener {...

public class HandleButtonEditMessageOnClick implements OnClickListener {...

public class HandleButtonEditSendToOnClick implements OnClickListener {

public static final String CLASS\_TAG = "HandleButtonEditSendToOnClick";

This is using the putExtra method that takes a Serializable object. If you look at SMSMessageData class it implements serializable. There is no putExtra method that takes a general Object type, it must be Serializable. You can see this aspect clearer in the code that uses the return result here.

public void onClick(View v) {

Log.i(CLASS\_TAG, "onClick started...");

Intent editIntent;

Activity sourceActivity;

Class destinationClass;

sourceActivity = GroupSMS.this;

destinationClass = EditSendTo.class;

editIntent = new Intent(sourceActivity, destinationClass);

// Sending information to the intent receiver through the Intent object

editIntent.putExtra("CURRENT\_PHONE", messageData);

startActivityForResult(editIntent, NEW\_SENDTO\_REQUEST);

}

}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

**super**.onActivityResult(requestCode, resultCode, data);

// Check which request we're responding to

**if** (requestCode == *NEW\_MESSAGE\_REQUEST*) {

// Make sure the request was successful

**if** (resultCode == *RESULT\_OK*) {

String newMessage = (String) (data.getStringExtra("NEW\_MESSAGE"));

messageData.setMessage(newMessage);

setSummary();

}

} **else** **if** (requestCode == *NEW\_SENDTO\_REQUEST*) {

// Make sure the request was successful

**if** (resultCode == *RESULT\_OK*) {

**this**.messageData = (SMSDataModel) (data.getSerializableExtra("NEW\_PHONE"));

setSummary();

}

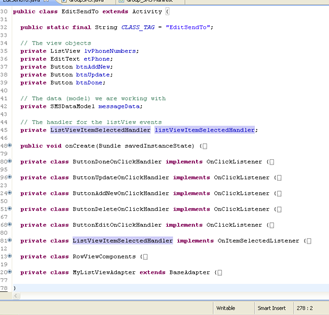
}

}

private void setSummary() {...

}

**EditSendTo.java (not all code is shown here, see download of files from Moodle)**



Note the use of named inner classes for the event handling

package itstudies.tafesa.groupSMS.views;

import itstudies.tafesa.groupSMS.R;...

public class EditSendTo extends Activity {

public static final String CLASS\_TAG = "EditSendTo";

// The view objects

Use the SMSDataModel passed through the intent

private ListView lvPhoneNumbers;

private EditText etPhone;

private Button btnAddNew;

private Button btnUpdate;

private Button btnDone;

// The data (model) we are working with

private SMSDataModelInterface messageData;

// The handler for the listView events

private ListViewItemSelectedHandler listViewItemSelectedHandler;

@Override

**public void onCreate(Bundle savedInstanceState) {**

super.onCreate(savedInstanceState);

Log.i(CLASS\_TAG, "onCreate ...");

setContentView(R.layout.activity\_edit\_send\_to);

// Get the SMSDataModel object that contains the data for this SMS Message

Intent editIntent;

editIntent = this.getIntent();

messageData = (SMSDataModelInterface) editIntent.getSerializableExtra("CURRENT\_PHONE");

if (messageData == null) {

messageData = new SMSDataModelList("");

}

// Set the Adapter for the ListView containing the list of Phone Numbers

lvPhoneNumbers = (ListView) this.findViewById(R.id.lvNumbers);

lvPhoneNumbers.setAdapter(new MyListViewAdapter(messageData));

// Setup the interface objects for this view

btnDone = (Button) this.findViewById(R.id.btnDonePhoneEdit);

btnUpdate = (Button) this.findViewById(R.id.btnUpdate);

btnAddNew = (Button) this.findViewById(R.id.btnAddNew);

etPhone = (EditText) this.findViewById(R.id.etPhone);

// Setup the event handlers for the ListView, Buttons and the EditText

listViewItemSelectedHandler = new ListViewItemSelectedHandler();

lvPhoneNumbers.setOnItemSelectedListener(listViewItemSelectedHandler);

lvPhoneNumbers.setOnItemClickListener(listViewItemSelectedHandler);

btnDone.setOnClickListener(new ButtonDoneOnClickHandler());

btnUpdate.setOnClickListener(new ButtonUpdateOnClickHandler());

btnAddNew.setOnClickListener(new ButtonAddNewOnClickHandler());

}

**private class ButtonDoneOnClickHandler implements OnClickListener {**

public static final String CLASS\_TAG = "ButtonDoneOnClickHandler";

public void onClick(View v) {

Log.i(CLASS\_TAG, "OnClick...");

Updated SMSDataModel containing the new phone numbers passed back through the activity result see GroupSMS .java onActivityResult method

Intent intent = new Intent();

intent.putExtra("NEW\_PHONE", messageData);

setResult(RESULT\_OK, intent);

finish();

}

}

**private class ButtonUpdateOnClickHandler implements OnClickListener {**

public static final String CLASS\_TAG = "ButtonUpdateOnClickHandler";

Our ListView handler will keep track of the position of the item we are working with

public void onClick(View v) {

Log.i(CLASS\_TAG, "OnClick...");

String oldNumber;

String newNumber;

int currentPositionInListView;

currentPositionInListView = listViewItemSelectedHandler.getCurrentSelectedPosition();

newNumber = etPhone.getText().toString();

if (newNumber.equals("")) {

Toast.makeText(EditSendTo.this,

"Cannot update to nothing. Use Delete to get rid of number!",

Toast.LENGTH\_LONG).show();

} else {

oldNumber = messageData.updatePhoneNumber(newNumber, currentPositionInListView);

Toast.makeText(

EditSendTo.this,"Update of item " + currentPositionInListView + " which was " + oldNumber + " to value "+ messageData.getPhoneNumber(currentPositionInListView), Toast.LENGTH\_LONG).show();

((MyListViewAdapter) lvPhoneNumbers.getAdapter()).notifyDataSetChanged();

}

This updates our data  
Important to get the ListView updated by calling the appropriate notify method

}

}

**private class ButtonAddNewOnClickHandler implements OnClickListener {**

//TODO – this one is your exercise

}

**private class ButtonDeleteOnClickHandler implements OnClickListener {**

//TODO – this one is your exercise

}

**private class ButtonEditOnClickHandler implements OnClickListener {**

public static final String CLASS\_TAG = "ButtonEditOnClickHandler";

**This is important**. This is how you find out which item/row in the ListView you are on when the EDIT button in the row is clicked

public void onClick(View v) {

int position;

View rowView;

rowView = (View) v.getParent();

position = lvPhoneNumbers.getPositionForView(rowView);

listViewItemSelectedHandler.onItemSelected((ListView) rowView.getParent(), rowView, position, position);

All the real work happens in the onItemSelected of the ListView handler. So we call that here

etPhone.requestFocus();

}

}

**private class ListViewItemSelectedHandler implements OnItemSelectedListener, OnItemClickListener {**

public static final String CLASS\_TAG = "ListViewItemSelectedHandler";

private int position = 0;

Keep track of the item position either selected or that has had a row button clicked.

public int getCurrentSelectedPosition() {

return position;

}

public void onItemSelected(AdapterView<?> parent, View rowView, int position, long id) {

RowViewComponents rowViewComponents;

rowViewComponents = (RowViewComponents) rowView.getTag();

This code updates the views at the top of the interface when we select a row in the List or click one of the row buttons - see onClick in ButtonEditOnClickHandler class)

EditText etPhone;

etPhone = (EditText) EditSendTo.this.findViewById(R.id.etPhone);

if (messageData.getPhoneNumber(position) != null) {

btnUpdate.setEnabled(true);

btnUpdate.setText("Update Entry " + (position + 1));

etPhone.setText(rowViewComponents.txtPhoneNumber.getText());

this.position = position;

} else {

btnUpdate.setEnabled(false);

rowViewComponents.btnDelete.setEnabled(false);

rowViewComponents.btnEdit.setEnabled(false);

}

}

**public** **void** onItemClick(AdapterView<?> parent, View view, **int** position, **long** id) {

onItemSelected(parent, view,position,id);

}

public void onNothingSelected(AdapterView<?> parent) {

// Nothing to do

}

The ListViews row layout has a tag with the rows view objects

}

private class RowViewComponents {

public TextView txtCounter;

public TextView txtPhoneNumber;

public Button btnEdit;

public Button btnDelete;

}

**private class MyListViewAdapter extends BaseAdapter {**

public static final String CLASS\_TAG = "MyListViewAdapter";

private SMSDataModelInterface theData;

Crucial method as it controls the ListView

This is used to access the data in the data model SMSDataModel.

public MyListViewAdapter(SMSDataModel theData) {

this.theData = theData;

}

public int getCount() {

return theData.getMaxNumPhoneNumbers();

}

public Object getItem(int position) {

return theData.getPhoneNumber(position);

}

public long getItemId(int position) {

return position;

}

public View getView(int position, View convertView, ViewGroup parent) {

View rowView;

RowViewComponents rowViewComponents;

if (convertView == null) {

LayoutInflater inflater = (LayoutInflater) (EditSendTo.this).getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

rowView = inflater.inflate(R.layout.phone\_number\_row\_layout, parent, false);

rowViewComponents = new RowViewComponents();

rowViewComponents.txtCounter = rowView.findViewById(R.id.txtCounter);

rowViewComponents.txtPhoneNumber = rowView.findViewById(R.id.txtPhone);

rowViewComponents.btnEdit = rowView.findViewById(R.id.btnEdit);

rowViewComponents.btnDelete = rowView.findViewById(R.id.btnDelete);

rowView.setTag(rowViewComponents);

// Register the handlers

rowViewComponents.btnEdit.setOnClickListener(new ButtonEditOnClickHandler());

rowViewComponents.btnDelete.setOnClickListener(new ButtonDeleteOnClickHandler());

//This allows clicking on the row in the ListView to be recognised

((ViewGroup)rowView).setDescendantFocusability(ViewGroup.*FOCUS\_BLOCK\_DESCENDANTS*);

} else {

rowView = convertView;

rowViewComponents = (RowViewComponents) rowView.getTag();

}

// Setup the TextViews showing the row count and phone number

//and enable/disable the buttons appropriately

rowViewComponents.txtCounter.setText("" + (position + 1));

Log.v(CLASS\_TAG ,"Updating position " + position + " value =" + getItem(position));

if (getItem(position) == null) {

rowViewComponents.txtPhoneNumber.setText("");

rowViewComponents.btnDelete.setEnabled(false);

rowViewComponents.btnEdit.setEnabled(false);

} else {

rowViewComponents.txtPhoneNumber.setText((String) getItem(position));

rowViewComponents.btnDelete.setEnabled(true);

rowViewComponents.btnEdit.setEnabled(true);

}

return rowView;

}

}

}