CSE3MAD LAB 3 – MULTIPLE ACTIVITIES

AIMS

- To explore further features of the android platform and Android studio.
- To build a multi-activity application.
- We understand you know Java, but you need to understand how Android projects are composed and the paradigms of developing on this platform.

BUILD A MESSAGING APPLICATION

Last week we developed an app by laying out widgets, using a layout control, responding to events. Let's go one step further!

Note: This is not a cut-and-paste exercise (hopefully you left that behind in first year), in fact it will not work as variable names will need to be synced and customised to your solution. The code however is instructive as to the steps you need to do to satisfy the goals of the app

Today's exercise is to build a simple text messaging app. The app will allow you to select a contact from your contact list, and send them an SMS

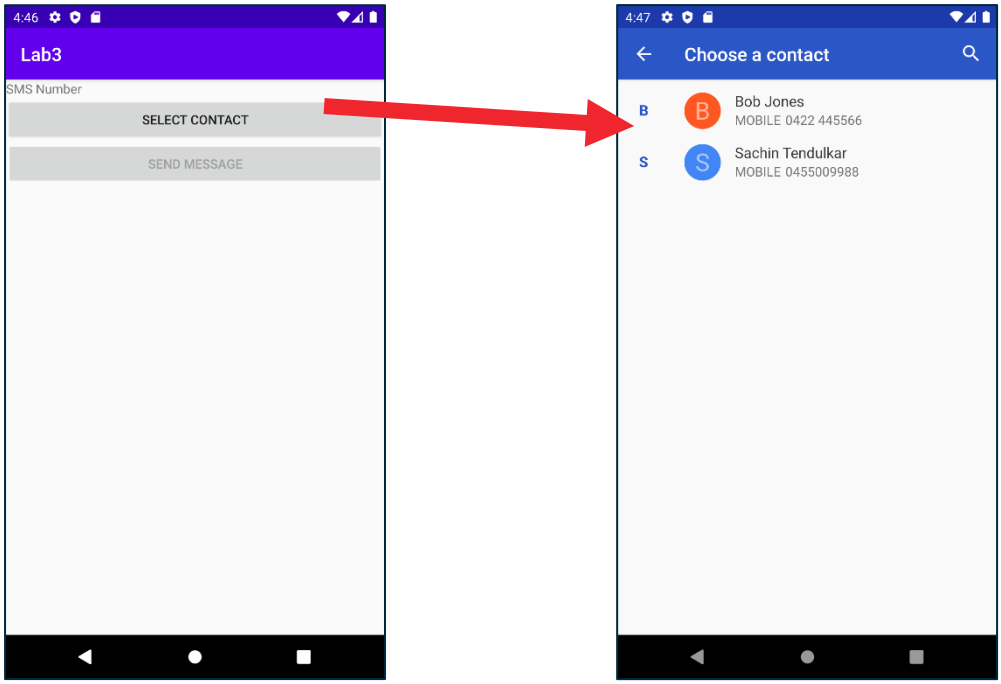


Figure 1

The launching activity (or the main activity) will have a button that opens the contact picker.

When the user selects a contact, the user will be taken back to the main activity & the selected phone number will be displayed & the ‘send message’ button will be enabled.

Figure 2

Figure 3

When the user clicks on the ‘send message’ button, a new activity is started. This second UI has an editable text input to type the message, and a button to send the SMS.

STEP 1 – STARTING THE PROJECT

a) Create a new project & choose the empty activity template. (See week 1 lab)
b) Create a new Android Project using API 27: Android Oreo
c) Create the UI in the visual DESIGN view for the main activity as in Figure 2.

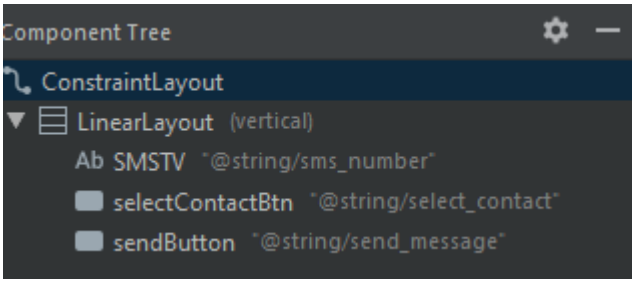


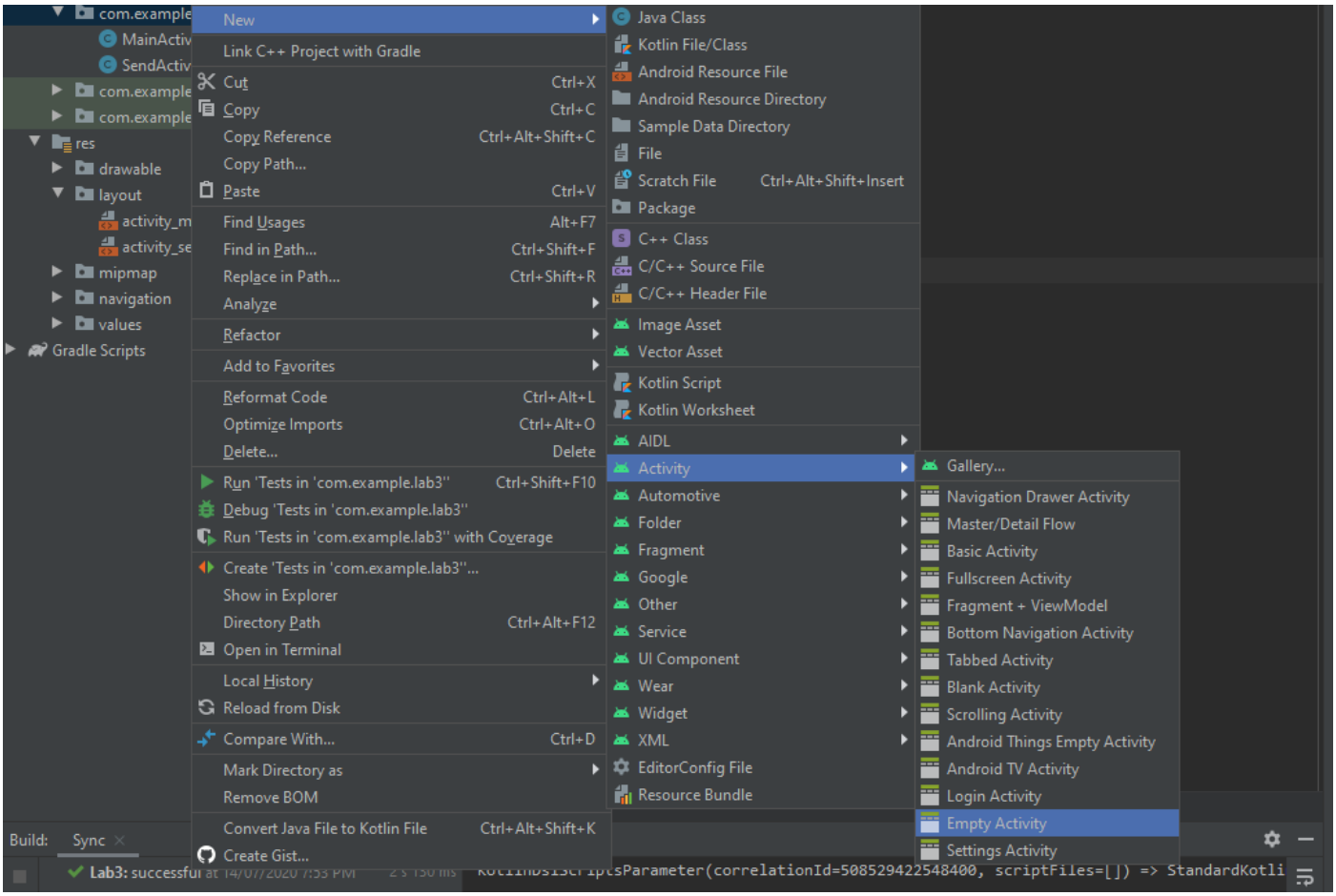
Figure 4

If you are stuck, here is my XML for the main activity layout, but remember it is actually easier to drag and drop from your palette into your component tree as per Figure 4!

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/SMSTV"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/sms_number" />

        <Button
            android:id="@+id/selectContactBtn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/select_contact" />

        <Button
            android:id="@+id/sendButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/send_message" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

STEP 2 – ADDING THE SENDACTIVITY

a) Create a new activity as per below and name it SendActivity.



b) Create the layout as seen in Figure 3. If you are stuck here is the XML.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SendActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <EditText
            android:id="@+id/smstext"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Msg..."
            android:gravity="start|top"
            android:inputType="textMultiLine"
            android:autofillHints="sms msg" />

        <Button
            android:id="@+id/sendMsgButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Send Text" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

STEP 3 – CODE THE MAIN ACTIVITY

a) Navigate in the IDE to the MainActivity.java file and add variables to access the UI elements in the activity. *Remember from last week where to declare and initialise them!* If you need a hand refer below.

```
Button selectButton = null;
Button sendButton = null;
TextView numView = null;

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


    selectButton = (Button) findViewById(R.id.selectContactBtn);
    sendButton = (Button) findViewById(R.id.sendButton);
    numView = (TextView) findViewById(R.id.SMSTV);
```

b) Add a listener for the select contact and the send message buttons.

```
//Button listener
selectButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});

sendButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

c) Now we need to add the necessary permissions to the AndroidManifest file

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.SEND_SMS" />
```

d) Because these are sensitive permission we need to check with the user at runtime that they will allow our app to have them. The first step is to make an array of the permissions we require, this is great if you require many permissions. You can put this up the top as a global variable.

```
//Required permissions array
final String[] PERMISSIONS = {Manifest.permission.READ_CONTACTS, Manifest.permission.SEND_SMS};
```

e) Create two new methods as below, basically the first methods checks of our app has permission to read the contacts and send sms messages on the mobile and the second will launch the launcher to ask for permissions during runtime. This is a very common pattern which we will examine later in the semester.

```
//helper function to check permission status
private boolean hasPermissions() {
    boolean permissionStatus = true;
    for (String permission : PERMISSIONS) {
        if (ActivityCompat.checkSelfPermission( context: this, permission) == PackageManager.PERMISSION_GRANTED) {
            Log.d(TAG,  msg: "Permission is granted: " + permission);
        } else {
            Log.d(TAG,  msg: "Permission is not granted: " + permission);
            permissionStatus = false;
        }
    }
    return permissionStatus;
}

//helper function to ask user permissions
private void askPermissions() {
    if (!hasPermissions()) {
        Log.d(TAG,  msg: "Launching multiple contract permission launcher for ALL required permissions");
        multiplePermissionActivityResultLauncher.launch(PERMISSIONS);
    } else {
        Log.d(TAG,  msg: "All permissions are already granted");
    }
}
```

f) Now we must make the Launcher to actually ask the required permissions, this can go beneath the 2 helper functions in the last question.

```
//Result launcher for permissions
private final ActivityResultLauncher<String[]> multiplePermissionActivityResultLauncher =
        registerForActivityResult(new ActivityResultContracts.RequestMultiplePermissions(), isGranted -> {
            Log.d(TAG,  msg: "Launcher result: " + isGranted.toString());
            if (isGranted.containsValue(false)) {
                Log.d(TAG,  msg: "At least one of the permissions was not granted, please enable permissions to ensure app functionality");
            }
        });
```

g) Now in the onCreate() we can call the askPermission() function to start the permission process.

```
//ask user for outstanding permissions
askPermissions();
```

h) Run the app. What happens when you click on the 'Select Contact' button? What happens when you click on the 'Send Message' button? It doesn't make sense to have the send button enabled before selecting a contact, so let's disable it in the onCreate method.

```
//make send button inactive
sendButton.setEnabled(false);
```

i) Now we need to complete the code inside select button. Here we do a final check to ensure permission are granted, if so we launch the contact picker launcher that we will code next.

```
//Button listener
selectButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //First we need to check permissions
        if (!hasPermissions()) {
            Log.d(TAG, msg: "Permission denied");
            Toast.makeText(getApplicationContext(), text: "Insufficent permissions to use app", Toast.LENGTH_SHORT).show();

        } else {
            //launch contact picker as we have permission
            Log.d(TAG, msg: "App has permission");
            mStartForResult.launch( input: null);
        }
    }
});
```

j)

k) Now we need to get the selected contact's phone number data into our app. First declare a member variable to store the phone number.

```
//Data variables
private String contactNumber = null;
```

l) Now let's add some code to get a contact, lets add this code under your onCreate() method.
In this code section we are using an ActivityResultLauncher (like when we asked permissions) to open the Contact picker and to aid us in querying the results when the user makes a selection.
We will be going though queries in greater depth later in the course.
If you find it hard to see you can copy it directly from this gist https://gist.github.com/latrobe-cs-educator/90bb0d6664a5327d6f942471456c37d5

NOTE: After we have retrieved the phone number we set its value in the numView so the user can see the number they picked. Additionally, now we have a number the set button is enabled once more.

```
ActivityResultLauncher<Void> mStartForResult = registerForActivityResult(
        new ActivityResultContracts.PickContact(), new ActivityResultCallback<Uri>() {
            @Override
            public void onActivityResult(Uri contactUri) {
                if(contactUri != null) {
                    Cursor cursor = getContentResolver().query(contactUri,
                            null,
                            null,
                            null,
                            null);
                    if (cursor != null && cursor.moveToFirst()) {
                        //First we get the user details
                        String contactId = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));
                        String contactName = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                        //Then we check if the user has a phone number to retrieve
                        String idResults = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER));
                        int idResultValue = Integer.parseInt(idResults);
                        Log.d(TAG, contactId + " " + contactName + " " + idResults);
                        cursor.close();
                        //If the user had a phone number we can then retrieve it
                        if (idResultValue == 1) // if the user has a phone number result is 1
                        {
                            //create a new cursor to run query that checks if selected Contact has a phone number
                            Cursor cursor2 = getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                                    null,
                                    ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " + contactId,
                                    null,
                                    null
                            );
                            //a contact may have multiple phone numbers currently we are just
                            // selecting the last one but you could add condition per requirement
                            while (cursor2.moveToNext()) {
                                //get phone number
                                contactNumber = cursor2.getString(cursor2.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
                                Log.d(TAG, contactNumber);
                                //set numView text to retrieved number
                                numView.setText(contactNumber);
                                //enable send button now that a phone number has been retrieved
                                if (contactNumber != null && contactNumber.length() > 0) {
                                    sendButton.setEnabled(true);
                                } else {
                                    sendButton.setEnabled(false);
                                }
                            }
                        }
                    }
                }
            }
        });
```

m) Guess what is next, we need to start the second activity, let's do this via a listener to the send button in the main activity. Note we are including the contact number data in the intent.

```
sendButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent myIntent = new Intent( packageContext: MainActivity.this, SendActivity.class);
        myIntent.putExtra( name: "contact_num", contactNumber);
        MainActivity.this.startActivity(myIntent);
    }
});
```

STEP 4 – CODE THE SEND ACTIVITY

a) Let's complete the second activity functionality. Open the SendActivity.java file. As before you are now experts in obtaining a reference to the UI widgets, for help see below.

```
//UX variables
private Button msgBtn = null;
private EditText msgText = null;
//Other
private String contact_number = null;
private String message = null;
static final int SMS_PERMISSION_REQ = 123; // PERMISSIONS VALUE
```

b) As before, initialise the members and this time the intent.

```
//Bind views
msgBtn = (Button) findViewById(R.id.sendMsgButton);
msgText = (EditText) findViewById(R.id.smstext);
//Get data from intent
Intent intent = getIntent();
contact_number = intent.getStringExtra( name: "contact_num");
```

c) Handle the event on the send message button of the second activity. (Hopefully you are getting an idea where to register listeners to UI widgets. **It's in the onCreate method**.

```
//Button listener
msgBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        sendMessage();
    }
});
```

d) Implement the sendMessage() method as called above. This can be outside the onCreate() as it is called from within the onCreate().

```
private void sendMessage() {
    message = msgText.getText().toString();
    SmsManager smsManager = SmsManager.getDefault();
    smsManager.sendTextMessage(contact_number,  scAddress: null, message,  sentIntent: null,  deliveryIntent: null);
    Toast.makeText( context: SendActivity.this,  text: "Message Sent", Toast.LENGTH_LONG).show();
}
```

RUN AND START TEXTING YOUR FRIENDS .....