What’s Your Story?

Developed By: Michelle Fiesta, Chia-Hui Hsieh, Alice Rhee, Stephanie Wooten

Have you ever wanted to connect with someone with a particular skill or area of interest, but had a hard time finding them? Have you ever yearned to connect with female engineers in your area, professional mixologists, a person with life experiences completely different from your own, or whose life has traveled a path you are interested in?

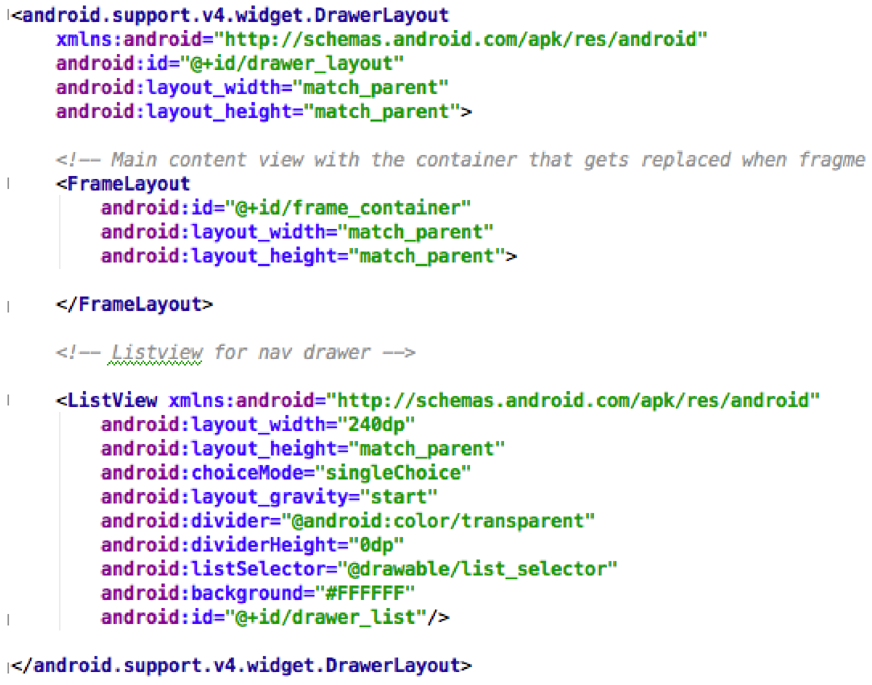
What’s Your Story? is about one-on-one connection on your own terms. It’s about real stories and life paths, and the chance to share your story with someone who is interested in *you*. It’s about getting questions answered that Google can’t and that LinkedIn freezes you out of. It’s about facilitating the reaching out process, about sharing knowledge and sharing pieces of yourself, about “I wish someone had told me this,” or “This isn’t so important after all.” It’s about coffee, lunch, an after-work drink at the pub—*What’s Your Story?* is an app about connecting, giving, and growing!

Our user groups are young adults 18+ and people with an interest giving back. These are generative people (35-50+) who want to be able to share a piece of themselves and their stories. Our app’s importance lies in bridging the connection of these two groups in meaningful and valuable ways. It’s an informational interview without the formality, bringing together people of different backgrounds who wouldn’t normally be able to connect.

Our two biggest competitors are LinkedIn and MeetUp, both sites used for connections and meetings aligned along specific interests. However, LinkedIn has a cold, professional ethos that can be impervious to those without pre-existing connections. MeetUps are group-based, not one-on-one, and still leave the step of further one-on-one meet up awkward. *What’s Your Story?* fills the gaps these two leave open. It is an app that promotes warm, friendly, low-pressure connection solely intended for the sharing of knowledge, wisdom, and personal stories.

Stephanie Wooten
Side Nav Drawer

First, I created the necessary layout xml files, including the layout for HomeActivity (activity_feed.xml), the fragment layout for the feed (fragment_feed.xml), adding the string array of nav drawer item titles to the strings.xml, and the individual layouts for the nav drawer items (listitem_in_nav_drawer.xml). The most important element of these layouts was adding a FrameLayout as the *first* layout within activity_feed.xml (Figure 1). The FrameLayout is the content view that is replaced as each fragment is called from the navigation drawer.



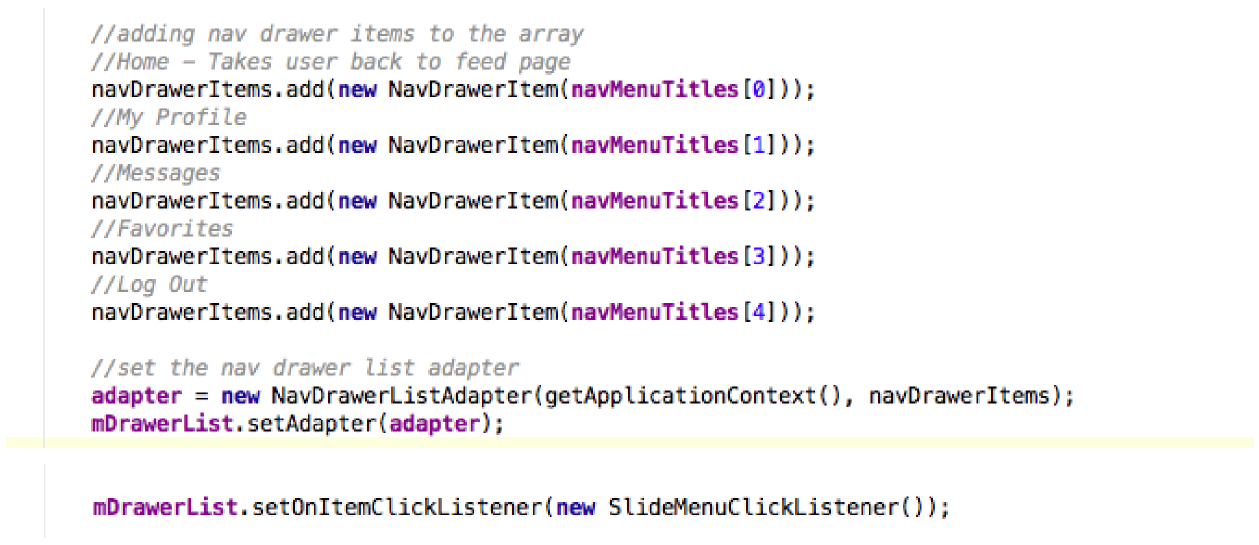
```xml
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Main content view with the container that gets replaced when fragme
    <FrameLayout
        android:id="@+id/frame_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    </FrameLayout>

    <!-- Listview for nav drawer -->

    <ListView xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:choiceMode="singleChoice"
        android:layout_gravity="start"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:listSelector="@drawable/list_selector"
        android:background="#FFFFFF"
        android:id="@+id/drawer_list"/>

</android.support.v4.widget.DrawerLayout>
```

Figure 1 - activity_feed.xml

Next, I built a custom list adapter for the navigation drawer list. The NavDrawerItem class contains the field and methods necessary to instantiate each list item object in the navigation drawer. For our nav drawer, I only declared a String field with the name, "title," and appropriate methods to "get" and "set" the title for each nav drawer item object. The NavDrawerListAdapter subclass inherits fields and methods from the Android BaseAdapter class. I declared two fields (context and the ArrayList of NavDrawerItems) and four "get" methods to this subclass. Within this subclass, the NavDrawerListAdapter constructor takes the two previously declared fields within it's parameters. This constructor is used in the HomeActivity subclass to create a NavDrawerListAdapter object - our navigation drawer "adapter".

The HomeActivity subclass inherits fields and methods from the Android Activity class. Within this subclass, I declared eight fields (one which is used within the onCreate method to instantiate the "adapter" object) and eight methods for the navigation drawer. Two important methods for making the side navigation drawer are the onCreate and displayView methods.

The onCreate method instantiates a number of objects, including NavDrawerItems needed for the list, a NavDrawerListAdapter object ("adapter"), and a SlideMenuClickListener object (see Figure 2). The displayView method contains a switch case that changes the activity screen dependent on which position is selected in the navigation drawer. Most switch cases in our displayView method contain intents that call other activity classes, with the exception of the default switch case which creates an instance of the FeedFragment subclass. Finally, the SlideMenuClickListener class (built within the HomeActivity subclass) implements the ListView.onItemClickListener interface. It sets the onclick listener for the navigation drawer, which invokes the displayView method to change the layout upon clicking certain list item positions within the navigation drawer list object.



```
//adding nav drawer items to the array
//Home - Takes user back to feed page
navDrawerItems.add(new NavDrawerItem(navMenuTitles[0]));
//My Profile
navDrawerItems.add(new NavDrawerItem(navMenuTitles[1]));
//Messages
navDrawerItems.add(new NavDrawerItem(navMenuTitles[2]));
//Favorites
navDrawerItems.add(new NavDrawerItem(navMenuTitles[3]));
//Log Out
navDrawerItems.add(new NavDrawerItem(navMenuTitles[4]));

//set the nav drawer list adapter
adapter = new NavDrawerListAdapter(getApplicationContext(), navDrawerItems);
mDrawerList.setAdapter(adapter);

mDrawerList.setOnItemClickListener(new SlideMenuClickListener());
```

Figure 2 - Portions of the onCreate method within HomeActivity subclass

The FeedFragment subclass, which inherits from the Android Fragment class, is used to create the only fragment object from which the user can view the navigation drawer. Since the navigation drawer overlays another screen layout, it must be created in association with a fragment object. This FeedFragment is the default view for the HomeActivity subclass (as is determined in the onCreate method for HomeActivity), and is overlayed when the nav drawer is called. It contains the onCreateView method, which creates an instance of the MySimpleArrayAdapter subclass (feedAdapter) and sets the on item click listener for that object. This allows users to select and view other user profiles from our feed list.

Chia Hui, Hsieh
Function: ListView and list data sending

I created a ListView with image and text that could show users a list of people profiles. Also, the data of each list item is sent to its detail page while users tap on it, including the image.

1) Create list related XML files:

First, I created view template *(listitem_in_activity_feed.xml)* for each item, including ImageView and TextView.

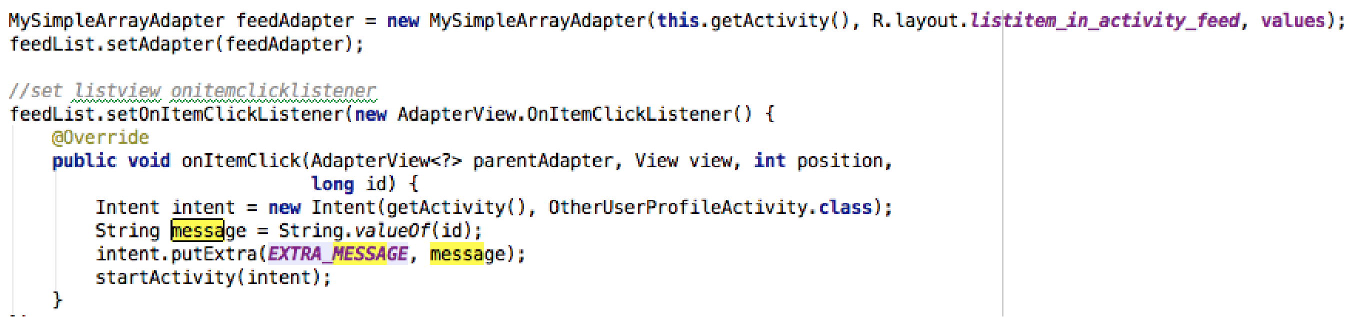
Then, I created another XML file for defining the ListView for each item *(fragment_feed.xml).* On this page, I also added a title with an icon and background color to tell users that this page is the list of people you might be interested in.

2) Create list related Java classes:

First, I created a User Data file *(FeedUserData)* for defining a model, given a Java object that has certain fields defined the FeedUserData class. This class contains a constructor method that using “this” to refer to the name, career, location and imagePath objects.

Second, I created an adapter class *(MySimpleArrayAdapter)* to describe the process of converting the feed user data to a view. I wrote a getView method to fetch the corresponding views. The method includes codes like inflating rows, getting references to TextView and ImageView by using findViewById, and setting data for view by using .setText. For image setting, I check whether the imagePath equals to null or not. If the imagePath is not null, then using setImageBitmap and BitmapFactory to get the image I stored in the asset folder before.

Third, I created a class to set the adapter for feed’s listview *(FeedFragment)*. I added several user data items *(initList)* to adapter first, then created an adapter *(feedAdapter)* and used .setAdapter to associate the list variables and its content with the list view. After that, set a ListView onItemClickListener which using .putExtra to pass the valueOf(id) message to the detail page. (*see figure 1*)

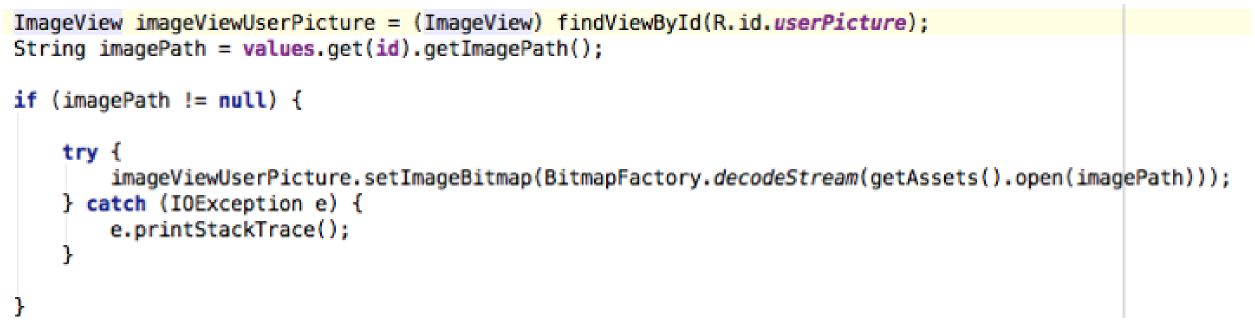


```
MySimpleArrayAdapter feedAdapter = new MySimpleArrayAdapter(this.getActivity(), R.layout.listitem_in_activity_feed, values);
feedList.setAdapter(feedAdapter);

//set listview onitemclicklistener
feedList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parentAdapter, View view, int position,
                            long id) {
        Intent intent = new Intent(getActivity(), OtherUserProfileActivity.class);
        String message = String.valueOf(id);
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
```

figure 1: Use adapter to associate list variable and its list view, then set ListView onItemClickListenter

3) Get list data:

On detail page (OtherUserProfile), I used getIntent and .getStringExtra to get the message of number of which item users tapped on the List. Then I created related TextView and ImageView to get references, then set the text content by using setText with the corresponding id; and set the image content by creating a string object “imagepath” for getting the id of the imagePath, then using the .setImageBitmap and BitmapFactoryto get the image *(See figure 2)*.



```java
ImageView imageViewUserPicture = (ImageView) findViewById(R.id.userPicture);
String imagePath = values.get(id).getImagePath();

if (imagePath != null) {

    try {
        imageViewUserPicture.setImageBitmap(BitmapFactory.decodeStream(getAssets().open(imagePath)));
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

Figure 2: get image content by get() and setImageBitmap

Alice Rhee

Login Verification

One functionality that I implemented in our app was the Log In verification. For this functionality, we had to ensure that the user could not just click "Log In" and enter the app without being prompted to actually log in with a correct username/password combination or sign up. I followed this tutorial (http://www.androidhive.info/2012/08/android-session-management-using-shared-preferences/) and modified it slightly to fit our app's needs.

A high-level overview of what needed to be done was that we needed to check to see 1) whether the user had input anything in the username and password fields at all, 2) whether what the user had input matched what was in our theoretical database and send them to the Home screen if so, and 3) set up Alert Dialogs that would give feedback to the user.

First, I created a new class called AlertDialogManager that contained a single method called showAlertDialog with 4 parameters: the context, the title of the dialog, the message, and a Boolean status (used to set an icon). Within the method, I built the alert dialog, set the title, message, and button OnClickListener, and told it to show up. Setting up this class with all of these Alert Dialog basics meant that later on I would not have to rewrite the code to build for each alert dialog from scratch, but could simply instantiate a new AlertDialogManager object and invoke that method with my situation-specific parameters.

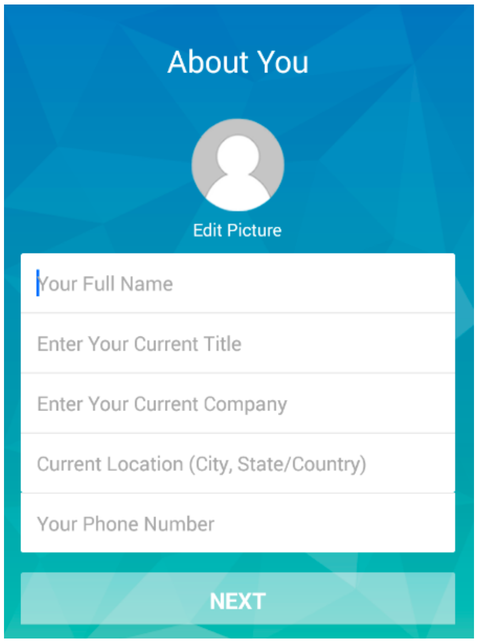
Second, I went into our LogInActivity and did some basic set up to link the critical UI elements in our XML to this Java file. In this case, the three UI elements that mattered were our username/password EditTexts and the Log In button. I declared these as variables in the Java file. Within the onCreate method, I then linked them to their corresponding XML elements using the findViewById method. Finally, I took the text the user had input into the EditTexts, converted them to strings using getText().toString(), and assigned them to a new string variable.

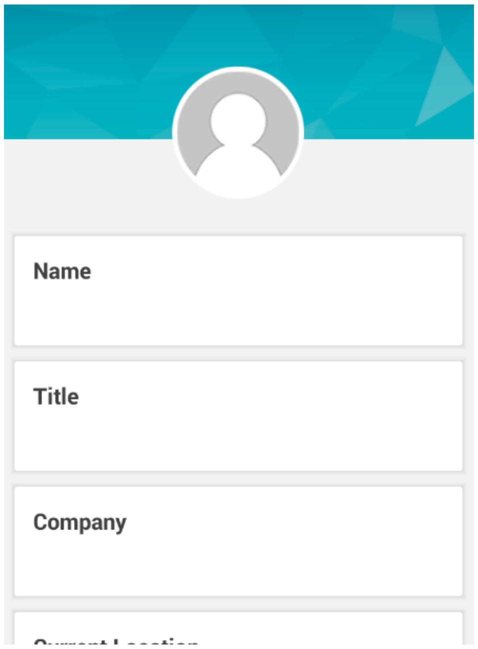
Third, I set up on onClickListener method for the Log In button that had a nested conditional. The first conditional checked the new string variables to see whether or not they had a length greater than 0 (had the user input anything at all?) using the built-in trim() and length() methods. If not, I called the AlertDialogManager object and asked it to show with the message: "Please enter username and password."

If the user had entered anything at all, then we went through to the next conditional, which checked to see if the username/password the user put in matched an accepted combination. I checked if the strings were the same through the equals() method. If the strings matched, then an intent to go to the HomeActivity was launched. If not, then another Alert Dialog was called telling the user that the "username/password is incorrect."

Michelle Fiesta
SharedPreferences - MyInitProfileActivity.java

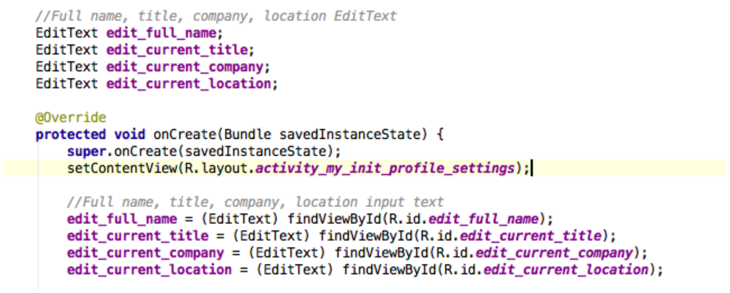
One of the functionalities that I was able to implement in the app was SharedPreferences for when the user first walks through the app during onboarding. They have the opportunity to fill out their profile information on the “About You” screen (MyInitProfileActivity.java), as shown in Figure 1. We wanted this information to then show up on the user’s personal profile page (MyProfile.java), shown in Figure 2, which is accessed via the Nav Drawer on the Feed Page.





In order to implement this feature, I first went in to the activity_my_init_profile_settings.xml document, which is the XML layout for the MyInitProfileActivity.java page. I created IDs for the four EditText views that we wanted to grab user inputted information from: edit_full_name, edit_current_title, edit_current_company, and edit_current_location.

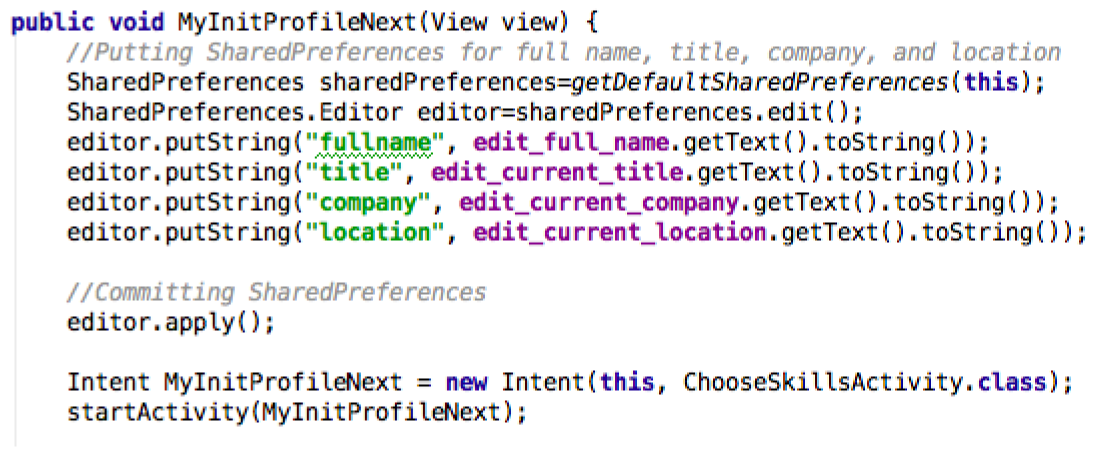
Next, in the onCreate method of the MyInitProfileActivity.java class, I declared four variables of the same names that I set equal to the views in the XML document.



```java
//Full name, title, company, location EditText
EditText edit_full_name;
EditText edit_current_title;
EditText edit_current_company;
EditText edit_current_location;

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

    //Full name, title, company, location input text
    edit_full_name = (EditText) findViewById(R.id.edit_full_name);
    edit_current_title = (EditText) findViewById(R.id.edit_current_title);
    edit_current_company = (EditText) findViewById(R.id.edit_current_company);
    edit_current_location = (EditText) findViewById(R.id.edit_current_location);
```

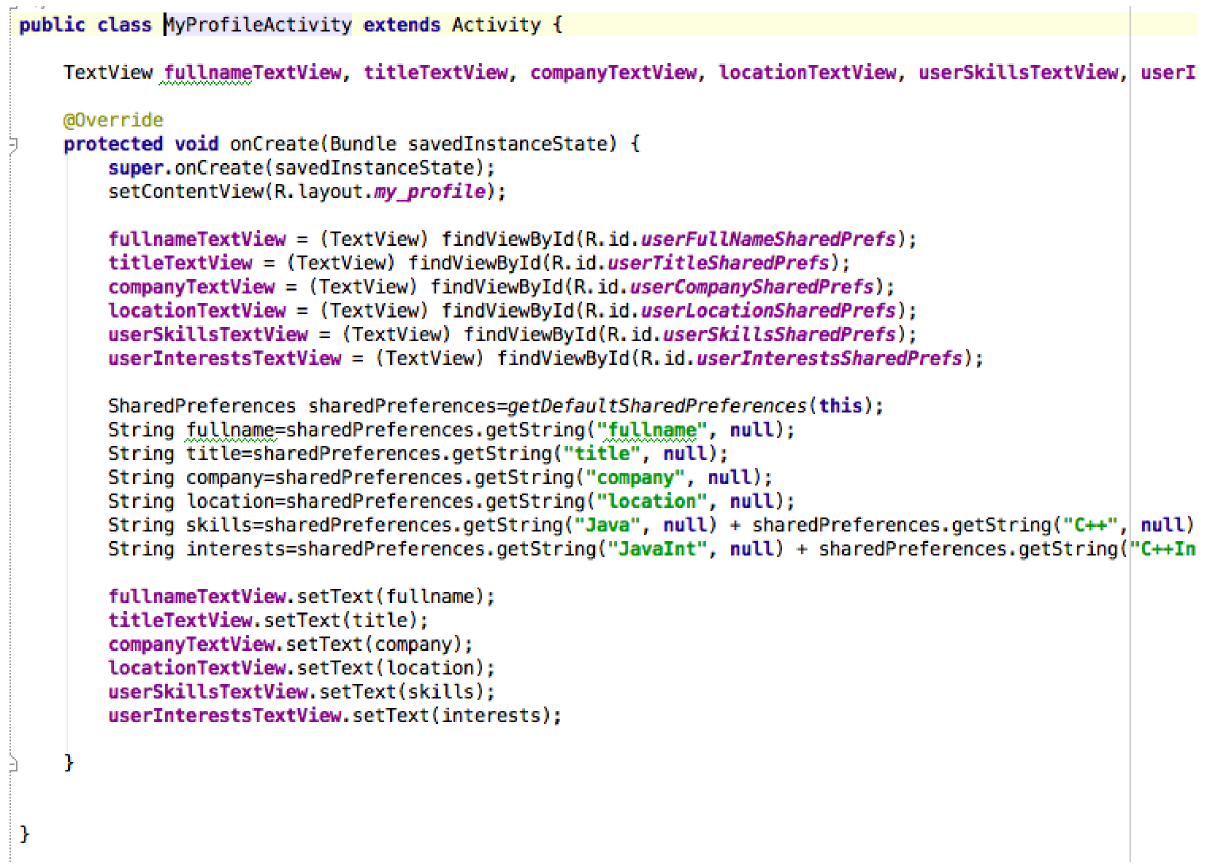
I later call these variables in the MyInitProfileNext method, which is called when the user clicks the Next button on the About You page.

I start by instantiating an object of the SharedPreferences method. I then create an editor object to in order to put the strings from the EditText views. I name the keys in a way that will be memorable for me, and then set the values equal to the EditText ID, transforming the data into texts and then into strings for storage.



```java
public void MyInitProfileNext(View view) {
    //Putting SharedPreferences for full name, title, company, and location
    SharedPreferences sharedPreferences=getDefaultSharedPreferences(this);
    SharedPreferences.Editor editor=sharedPreferences.edit();
    editor.putString("fullname", edit_full_name.getText().toString());
    editor.putString("title", edit_current_title.getText().toString());
    editor.putString("company", edit_current_company.getText().toString());
    editor.putString("location", edit_current_location.getText().toString());

    //Committing SharedPreferences
    editor.apply();

    Intent MyInitProfileNext = new Intent(this, ChooseSkillsActivity.class);
    startActivity(MyInitProfileNext);
```

Finally, in the MyProfileActivity.java activity, I call the variables of the TextViews that I want to display my values in: userFullNameSharedPrefs, userTitleSharedPrefs, userCompanySharedPrefs, userLocationSharedPrefs. These match the IDs for the view objects in the XML for the "My Profile" page. I use SharedPreferences again, except instead of putting strings I am getting the strings. I supply the keys for the values that I want to display, and set the default to "null" in the case that the user did not choose to fill out that section.



```java
public class MyProfileActivity extends Activity {

    TextView fullnameTextView, titleTextView, companyTextView, locationTextView, userSkillsTextView, userI

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

        fullnameTextView = (TextView) findViewById(R.id.userFullNameSharedPrefs);
        titleTextView = (TextView) findViewById(R.id.userTitleSharedPrefs);
        companyTextView = (TextView) findViewById(R.id.userCompanySharedPrefs);
        locationTextView = (TextView) findViewById(R.id.userLocationSharedPrefs);
        userSkillsTextView = (TextView) findViewById(R.id.userSkillsSharedPrefs);
        userInterestsTextView = (TextView) findViewById(R.id.userInterestsSharedPrefs);

        SharedPreferences sharedPreferences=getDefaultSharedPreferences(this);
        String fullname=sharedPreferences.getString("fullname", null);
        String title=sharedPreferences.getString("title", null);
        String company=sharedPreferences.getString("company", null);
        String location=sharedPreferences.getString("location", null);
        String skills=sharedPreferences.getString("Java", null) + sharedPreferences.getString("C++", null)
        String interests=sharedPreferences.getString("JavaInt", null) + sharedPreferences.getString("C++In

        fullnameTextView.setText(fullname);
        titleTextView.setText(title);
        companyTextView.setText(company);
        locationTextView.setText(location);
        userSkillsTextView.setText(skills);
        userInterestsTextView.setText(interests);

    }


}
```