**Grow with Google Challenge Scholarship**

**Lesson 7 Notes**

**Edit Text Preference Constraints - Setting an acceptable range**

To limit the acceptable values between 0 (non inclusive) and 3 (inclusive) we opted to use a PreferenceChangeListener - this is not the same as a SharedPreferenceChangeListener. The differences are:

* SharedPreferenceChangeListener is triggered after any value is saved to the SharedPreferences file.
* PreferenceChangeListener is triggered before a value is saved to the SharedPreferences file. Because of this, it can prevent an invalid update to a preference. PreferenceChangeListeners are also attached to a single preference.

Generally the flow goes like this:

1. User updates a preference.
2. PreferenceChangeListener triggered for that preference.
3. The new value is saved to the SharedPreference file.
4. onSharedPreferenceChanged listeners are triggered.

Otherwise they act very similarly. In your activity you implement the Preference.OnPreferenceChangeListener, override the onPreferenceChange(Preference preference, Object newValue). The onPreferenceChange method will return either true or false, depending on whether the preference should actually be saved.

Okay, so let’s take a look at the code.

**Step 1: Implement the OnPreferenceChangeListener:**

public class SettingsFragment extends PreferenceFragmentCompat implements OnSharedPreferenceChangeListener, Preference.OnPreferenceChangeListener

**Step 2: Attach the listener to the size preference in onCreatePreferences**

@Override

public void onCreatePreferences(Bundle bundle, String s) {

/\* Other preference setup code code \*/

//...

Preference preference = findPreference(getString(R.string.pref\_size\_key));

preference.setOnPreferenceChangeListener(this);

}

**Step 3: Implement onPreferenceChange**

This code will first try to convert the preference into a number and then checks that the number is between 0 and 3. If either of these fail, a toast will be show and false is returned. By returning false, the incorrect value is not saved to shared preferences.

public boolean onPreferenceChange(Preference preference, Object newValue) {

Toast error = Toast.makeText(getContext(), "Please select a number between 0.1 and 3", Toast.LENGTH\_SHORT);

String sizeKey = getString(R.string.pref\_size\_key);

if (preference.getKey().equals(sizeKey)) {

String stringSize = (String) newValue;

try {

float size = Float.parseFloat(stringSize);

if (size > 3 || size <= 0) {

error.show();

return false;

}

} catch (NumberFormatException nfe) {

error.show();

return false;

}

}

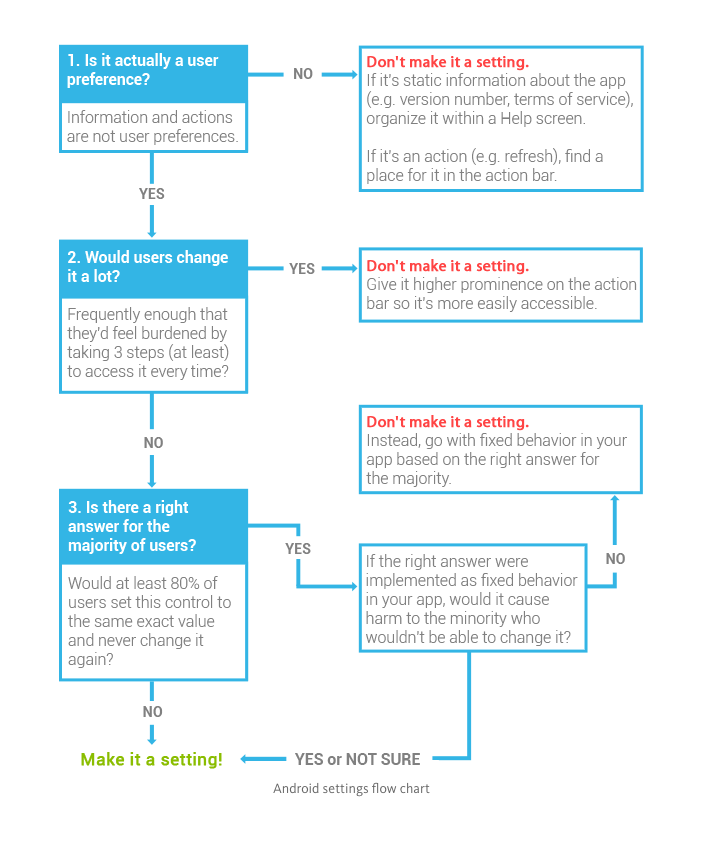
return true;

}

**Should it be a Setting?**

Material Design Guidelines are the new standard for anything android, [**here**](https://material.google.com/patterns/settings.html)'s a link to the guidelines for how to create settings pages

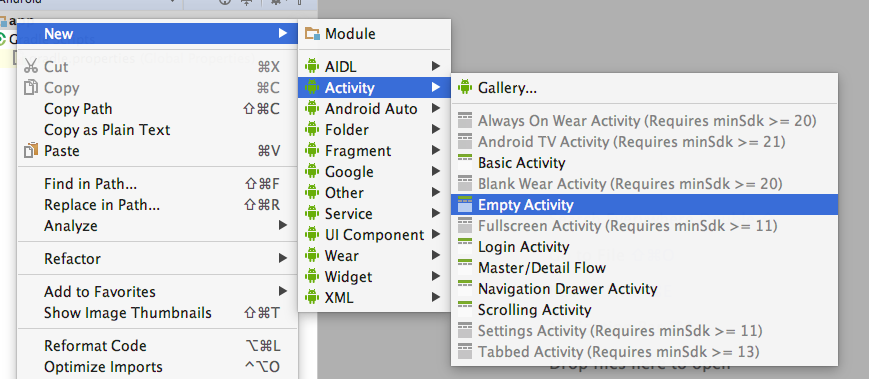
Here's a handy flowchart to help you decide if something is worth being a settings or should you just decide for the user a default value for:



# Create the SettingsActivity in Sunshine Solution

## Notes on Solution Code

First you should create the activity. That can be done by going to **New** > **Activity** > **Empty Activity** as seen below. It's important to use **Empty Activity** and not **Basic Activity** because Basic Activity actually contains some UI elements you don't want.

**[[](https://classroom.udacity.com/courses/ud851-gwg/lessons/1392b674-18b6-4636-b36b-da7d37a319e3/concepts/3b94732f-bd75-4da5-8f7f-52b67af8160f)](https://classroom.udacity.com/courses/ud851-gwg/lessons/1392b674-18b6-4636-b36b-da7d37a319e3/concepts/3b94732f-bd75-4da5-8f7f-52b67af8160f)**

One you have the activity, you need to add menu items to open the activity. To do this, you'll add the xml for the menu item in **detail.xml** and **forecast.xml** in the menu folder. This makes the menu item appear, but doesn't make it do anything. To make it actually open the SettingsActivity you need to go to the onOptionsItemSelected in both MainActivity and DetailActivity and write an explicit intent to start the SettingsActivity:

**if** (id == R.id.action\_settings) {

Intent startSettingsActivity = **new** Intent(**this**, SettingsActivity.class);

startActivity(startSettingsActivity);

**return** **true**;

}

There are a few more small but important notes. Use android:launchMode="singleTop" to make sure that a new instance of MainActivity isn't spawned when you press the back button. Spawning a new instance of MainActivity is not memory efficient.

Then, in the actual SettingsActivity you should override the home button to act like the back button:

**if** (id == android.R.id.home) {

onBackPressed();

}

and you should display home as up, to [**allow up navigation**](https://developer.android.com/training/implementing-navigation/ancestral.html):

**this**.getSupportActionBar().setDisplayHomeAsUpEnabled(**true**);

### Solution Code

**Solution:** [[**S06.01-Solution-LaunchSettingsActivity**](https://github.com/udacity/ud851-Sunshine/tree/student/S06.01-Solution-LaunchSettingsActivity)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S06.01-Exercise-LaunchSettingsActivity...S06.01-Solution-LaunchSettingsActivity)]