

Lab 3: Jersey (Menus & Dialogs)

Goal

In this lab, we'll get practice with menus and dialogs by creating an app that displays basketball jerseys with customizable names, numbers, and colors. We will also learn about localizing strings for international deployment and about persisting data using SharedPreferences.

[Requirements](#)

[Screenshots/Testing](#)

[Hints/guidelines](#)

Requirements


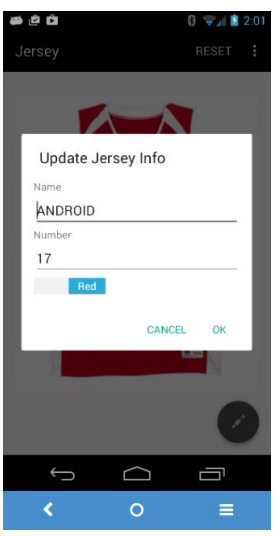
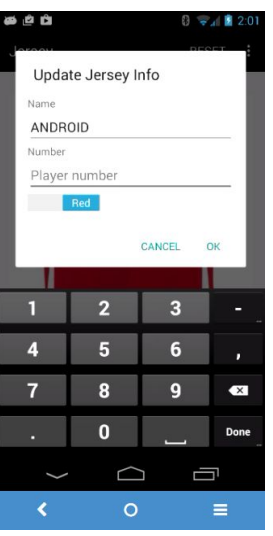

1. You must use a model object for the Jersey info. It contains a player name (String), player number (int), and if the jersey is red (boolean).
2. When the app starts for the first time, a red jersey appears with the name ANDROID and the number 17.
3. The user can launch a dialog to edit the jersey info by pressing the gray floating action button at the bottom right. This launches a dialog that auto-fills the fields with the current values on the jersey: player name (String), player number (int), and is jersey red (boolean).
4. Clicking OK or Cancel on the edit activity goes back to the activity. OK will pass the entered data back, but cancel won't.
5. The user can only put numbers for the jersey number; no other text is allowed. (If they try to input non numbers or leave it blank, just set the number to 0 and don't crash the app.)
6. The user can reset the jersey info to its initial values by pressing reset in the menu item. When the user chooses this, it will show a dialog to confirm if the user really wants to reset all the data. It will only reset if the user presses OK. Alternatively, you can remove it and use the snackbar with UNDO to get it back.
7. Your app must look exactly like the screenshots if you are using a phone. (Exception: you don't need to use a switch; see note below.) If your image is resized on a tablet, feel free to change font sizes as needed on the jersey to display nicely. If you are on a tablet, some things might look small - you would use fragments to get it right, but we won't worry about that in this lab. You also don't need to worry about handling screen rotation.
8. Likewise, your dialog buttons should look like those shown. If you are testing on an older device, `import android.support.v7.app.AlertDialog`; to get the nice Material Design borderless button style on your AlertDialog.
9. Use string resources for all text displayed on the screen. Recall the alt-shift hint.
10. Use the given images. You can set the icon in the manifest: [Jersey Images](#)
11. When the user clicks settings, it will go to the System settings page for Language.
12. The app also has support for Brazilian Portuguese; that is, when that language is selected by the user, all the text should switch.
13. When the app is closed, the jersey info is saved in SharedPreferences. When it is relaunched, it reloads the jersey info from shared preferences and displays the

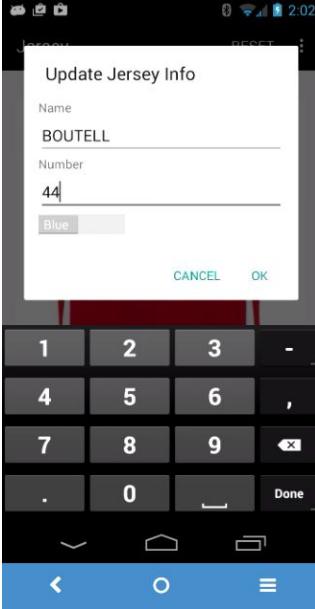

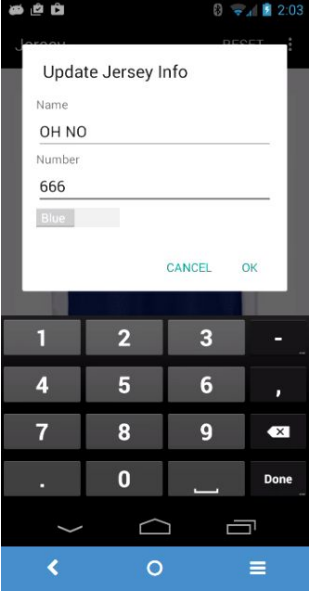

appropriately-labeled jersey on the screen.


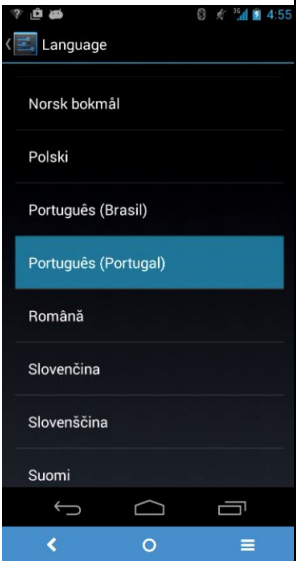
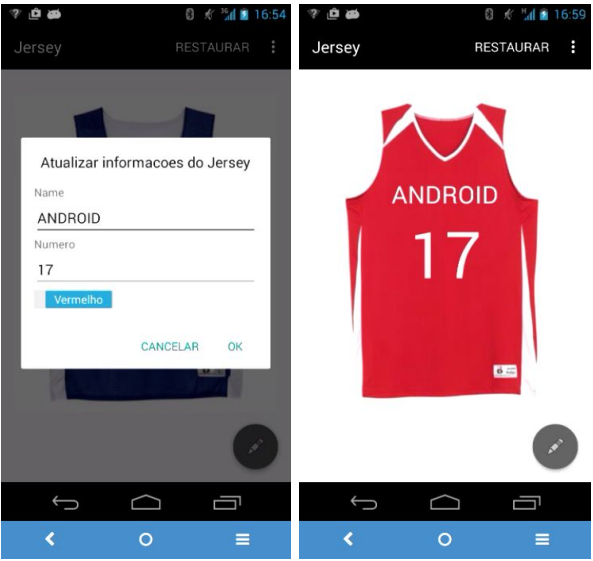
14. Partial credit: data persists, language settings, reset, edit dialog, all 2 pts each, & 2 pts for rest of app.

Screenshots/Testing

Here was most of my basic test :

Launch	Edit dialog	Change #to blank	OK Button doesn't crash
			

Change to Valid	OK Button!	Cancel ignores
		
		

After stop and restart app (jersey info persisted)	Settings, then choose Portuguese	In Portuguese. Go, strings.xml! Then after choosing Restaurar (Reset).
		

Hints/guidelines

1. I recommend you skip DialogFragments altogether and just use AlertDialogs! (If you were to insist on using DialogFragments, you'd need to set up your app to use version 23 (not the default of 26) when setting up the project in the module-level build.gradle).
2. Start with BasicActivity template, not EmptyActivity, to get FAB.
3. Strings.xml are below.
4. The FAB icon is `@android:drawable/ic_menu_edit`.
5. For the ImageView, you set the image in code by calling `setImageResource()`.
6. To get numeric input only for the number, you can set the keyboard type to numeric. Look up how to do this. As a fallback option, if you used a regular keyboard and the user types a bad value, just catch the `NumberFormatException` thrown by `Integer.parseInt()`.
7. To select the color, you may use a Switch, a Checkbox, or a ToggleButton (your choice).
<http://developer.android.com/guide/topics/ui/controls/togglebutton.html>
<http://developer.android.com/reference/android/widget/Switch.html>
<https://developer.android.com/guide/topics/ui/controls/checkbox.html>
8. A crash with message **Caused by:**
android.content.res.Resources\$NotFoundException: String resource ID #0x11
usually means that you are calling `setText()` with an integer (the jersey number), and since `setText()` is overloaded to expect resource IDs if you pass it an int, it will look for that resource and fail. Instead, pass in a string version of the int.

9. You can launch any system settings using an intent. There is an error in this (it's the `android.provider.Settings` class that has the various ACTIONS, not the Intent class), otherwise it is fine:
<http://developer.android.com/guide/components/intents-common.html#Settings>
10. It's fairly easy to support another language, by creating a new strings.xml file with the new values. Here is what I used. Read about it here:
<http://developer.android.com/training/basics/supporting-devices/languages.html>. If you get serious about this, here is more info:
<http://developer.android.com/guide/topics/resources/localization.html> You can google for the right language code. Spoiler: it's pt-br.
11. You can do this in Android Studio by making a new resource file (right click on the `res/values` folder and this should be the option). Type `string.xml`, and then choose locale `>>`, Language = Portuguese, Region = Brazil.

Here are some strings for you to use:

```
<resources>
  <string name="app_name">Jersey</string>
  <string name="edit">Edit</string>
  <string name="player_name_hint">Player name</string>
  <string name="number">Number</string>
  <string name="player_number_hint">Player number</string>
  <string name="blue">Blue</string>
  <string name="red">Red</string>
  <string name="action_settings">Settings</string>
  <string name="reset">Reset</string>
  <string name="confirmation_dialog_message">Reset jersey info?</string>
  <string name="edit_dialog_title">Update Jersey Info</string>
  <string name="default_jersey_name">ANDROID</string>
  <string name="name">Name</string>
</resources>
```

And in Portuguese for the new strings.xml file:

```
<resources>
  <string name="app_name">Jersey</string>
  <string name="edit">Editar</string>
  <string name="player_name_hint">Nome do jogador</string>
  <string name="number">Numero</string>
  <string name="player_number_hint">Numero do jogador</string>
  <string name="blue">Azul</string>
  <string name="red">Vermelho</string>
  <string name="action_settings">Configuracoes</string>
  <string name="reset">Restaurar</string>
  <string name="confirmation_dialog_message">Restaurar informacoes do
```

```

Jersey?</string>
    <string name="edit_dialog_title">Atualizar informacoes do Jersey</string>
    <string name="default_jersey_name">ANDROID</string>
    <string name="name">Nome</string>
</resources>

```

12. For later consideration: you can specify many resources based on the device or locale: Country code, text direction (for Arabic), screen sizes, densities, and aspect ratios, night vs. not (think GPS in car), touch, stylus, keyboard, ...

<http://developer.android.com/guide/topics/resources/providing-resources.html>

Accessing them happens automatically when the device type, time, or locale matches.

The key is testing your app on various devices (or at least emulators with those settings).

No action required now other than to test your app in both English and Portuguese.

13. SharedPreferences are used to persist data between app launches in a way that is simpler than full-scale SQLite or cloud storage, much less scalable. We'll use them to persist the fields of the jersey object. Steps are:

- a. **Saving data.** When an activity is paused, its onPause() method is called, and this is a good time to store data. It is done transactionally, so you get an editor, make a bunch of changes, then commit them. In each of these, I show how to store the player name; you'll do similarly for the number and color boolean.

// Constants:

```

private final static String PREFS = "PREFS";
private static final String KEY_JERSEY_NAME = "KEY_JERSEY_NAME";

```

// Add this method:

```

@Override
protected void onPause() {
    super.onPause();
    SharedPreferences prefs = getSharedPreferences(PREFS, MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString(KEY_JERSEY_NAME, mJersey.getPlayerName());
    // Put the other fields into the editor
    editor.commit();
}

```

- b. **Loading data.** You want to load in the saved values the next time the app is run. So add these to onCreate():

```

SharedPreferences prefs = getSharedPreferences(PREFS, MODE_PRIVATE);
String name =
prefs.getString(KEY_JERSEY_NAME, getString(R.string.default_jersey_name));
// Get the other fields. Then use them all

```

Credits

Thanks to exchange student Felix Jose Batista for providing the Portuguese translation!
And CSSE alumnus Eric Stokes made these images while a student at Rose.