Git & Dialogs Lab

# There are two portions to this lab. The first part must be done with the group members and the second part must be done individually.

1. Git and GitHub

You do need to work with other group members to complete this lab. However, your individual work will be graded.

*Repeat this portion till you feel comfortable with this process.*

(2 Points) For extra credit, you may choose to complete the Git tutorial at <https://www.codecademy.com/learn/learn-git>. Create a codecademy account, complete the tutorial and show the instructor that you completed in person. It’s best if you complete the tutorial before starting this lab.  
  
Other resources: <https://git-scm.com/book/en/v2>

Preconditions:

1. Git must be installed on your computer (if you haven’t in prior classes) otherwise it won’t work with Android Studio. Please do this prior to coming to class so that you don’t have to compete for network bandwidth.
2. Create a GitHub.com account using <https://help.github.com/articles/applying-for-a-student-developer-pack/>

First Group Member

1. Create a new Android project for testing with your group. (Only one of the group members should create this project and will act as the main contributor. This one group member will be in charge of merging the branches later. Discuss who will be the main contributor among your team members). It will be the default branch (master).
2. Make sure that you can build and run this project. Share this project on GitHub. Make sure that you can see all the files on GitHub.com in your account.
3. On GitHub.com, navigate to your project and create a new branch for each group member. Name the branch with their first name. Ideally, you should create a branch by issue or by the feature name or the bug name. This is how it is done in the industry.

Other Group Member(s)

1. Fork the branch from GitHub.com with your name. You can also do this from Android Studio.
2. Open it in Android by going through Git and select the corresponding branch and add something to your particular branch and commit the changes. Create a pull request on GitHub.com after completion.

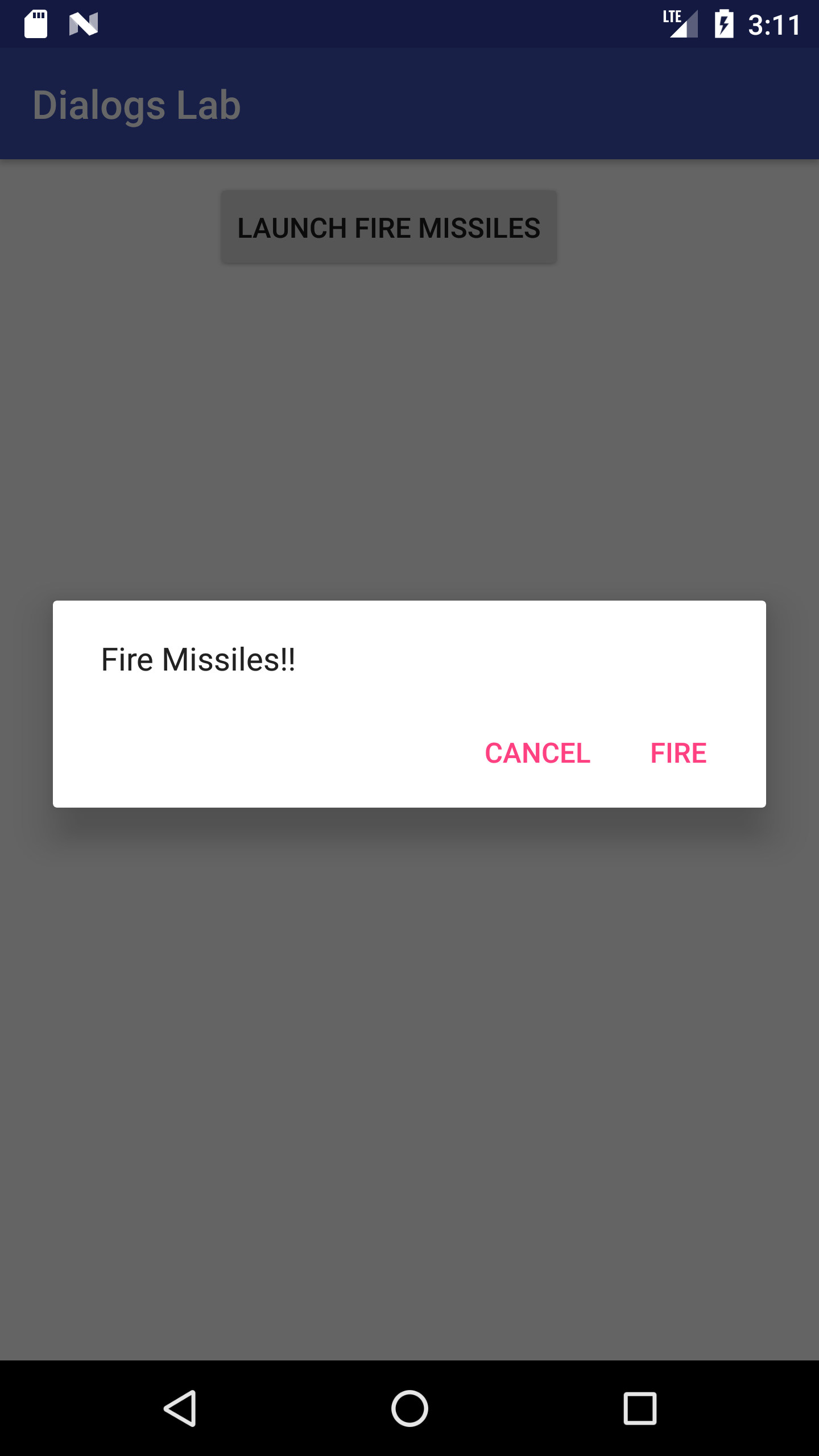
First Group Member

1. Merge the pull request into the branch and merge the branches into the master.
2. Open in Android to see if you can see all the changes.
3. Dialogs

Complete the following portion individually and show your work to the instructor in person before the deadline. You may also help others if you finish your work early. No email submissions will be accepted. It’s best to type in some of the code so that you can understand what’s going on.

1. Create a new Project named Dialogs Lab to practice dialogs. Create an Activity (Empty) that can be used to launch the different dialogs that we are going to practice. I named mine, DialogsActivity using Empty Activity template.

## Dialogs – Simple Ok, Cancel



1. Create a new Fragment class called FireMissilesDialogFragment without layout and callback methods using Blank Fragment template.

Create strings to use with the dialog in **strings.xml**.

**<string name="dialog\_fire\_missiles">Fire Missiles!!</string>  
<string name="fire">Fire</string>  
<string name="cancel">Cancel</string>**

**<!-- buttons -->**

**<string name="text\_file\_missiles">LAUNCH FIRE MISSILES</string>**

1. Modify the FireMissilesDialogFragment class to extend DialogFragment. Remove the onCreateView method. Use the Import from   
   import android.support.v4.app.DialogFragment;
2. Override onCreateDialog method to show the title, positive and negative buttons as shown in code below.

**@Override**

**public Dialog onCreateDialog(Bundle savedInstanceState) {**

**// Use the Builder class for convenient dialog construction**

**AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());**

**builder.setMessage(R.string.dialog\_fire\_missiles)**

**.setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int id) {**

**// FIRE ZE MISSILES!**

**}**

**})**

**.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int id) {**

**// User cancelled the dialog**

**}**

**});**

**// Create the AlertDialog object and return it**

**return builder.create();**

**}**

1. Change the xml layout file of the ***MainActivity***’s class to create a button to launch each type of fragment that we will create in this lab. Here’s what some of the attributes are assigned.

**android:id="@+id/btn\_fire\_missiles"  
android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content"  
android:onClick="launch"  
android:text="@string/text\_file\_missiles"**

1. Add the launch method to the ***MainActivity’s*** class.

**public void launch(View view) {  
  
 DialogFragment fragment = null;  
 if (view.getId() == R.id.*btn\_fire\_missiles*) {  
 fragment = new FireMissilesDialogFragment();  
 } if (fragment != null)  
 fragment.show(getSupportFragmentManager(), "launch");  
}**

1. Launch the app and see if it works.

**Dialogs – List**



1. Change the ***strings.xml*** file to add an array and title.

**<string name="pick\_color">Pick a Color</string>**

**<array name="colors\_array">**

**<item name="red">Red</item>**

**<item name="green">Green</item>**

**<item name="blue">Blue</item>**

**</array>  
 <string name="launch\_colors">Launch Colors</string>**

1. Create a new fragment called ***ListDialogFragment*** similar to the previous problem that extends ***DialogFragment***. Add the code to setItems of an array to show a list of items.

**public Dialog onCreateDialog(Bundle savedInstanceState) {**

**AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());**

**builder.setTitle(R.string.pick\_color)**

**.setItems(R.array.colors\_array, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int which) {**

**// The 'which' argument contains the index position**

**// of the selected item**

**Resources res = getActivity().getResources();**

**String[] colors = res.getStringArray(R.array.colors\_array);**

**Toast.makeText(getActivity(), "You chose " + colors[which], Toast.LENGTH\_SHORT)**

**.show();**

**}**

**});**

**return builder.create();**

**}**

1. Change the layout of ***MainActivity*** to add a new button to test this dialog.

**android:id="@+id/btn\_launch\_colors"  
android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content"  
android:layout\_margin="10dp"  
android:onClick="launch"  
android:text="@string/launch\_colors"**

1. Modify the ***launch*** method for the new button.

**public void launch(View view) {**

**DialogFragment fragment = null;**

**if (view.getId() == R.id.btn\_fire\_missiles) {**

**fragment = new FireMissilesDialogFragment();**

**} else if (view.getId() == R.id.btn\_launch\_colors) {**

**fragment = new ListDialogFragment();**

**}**

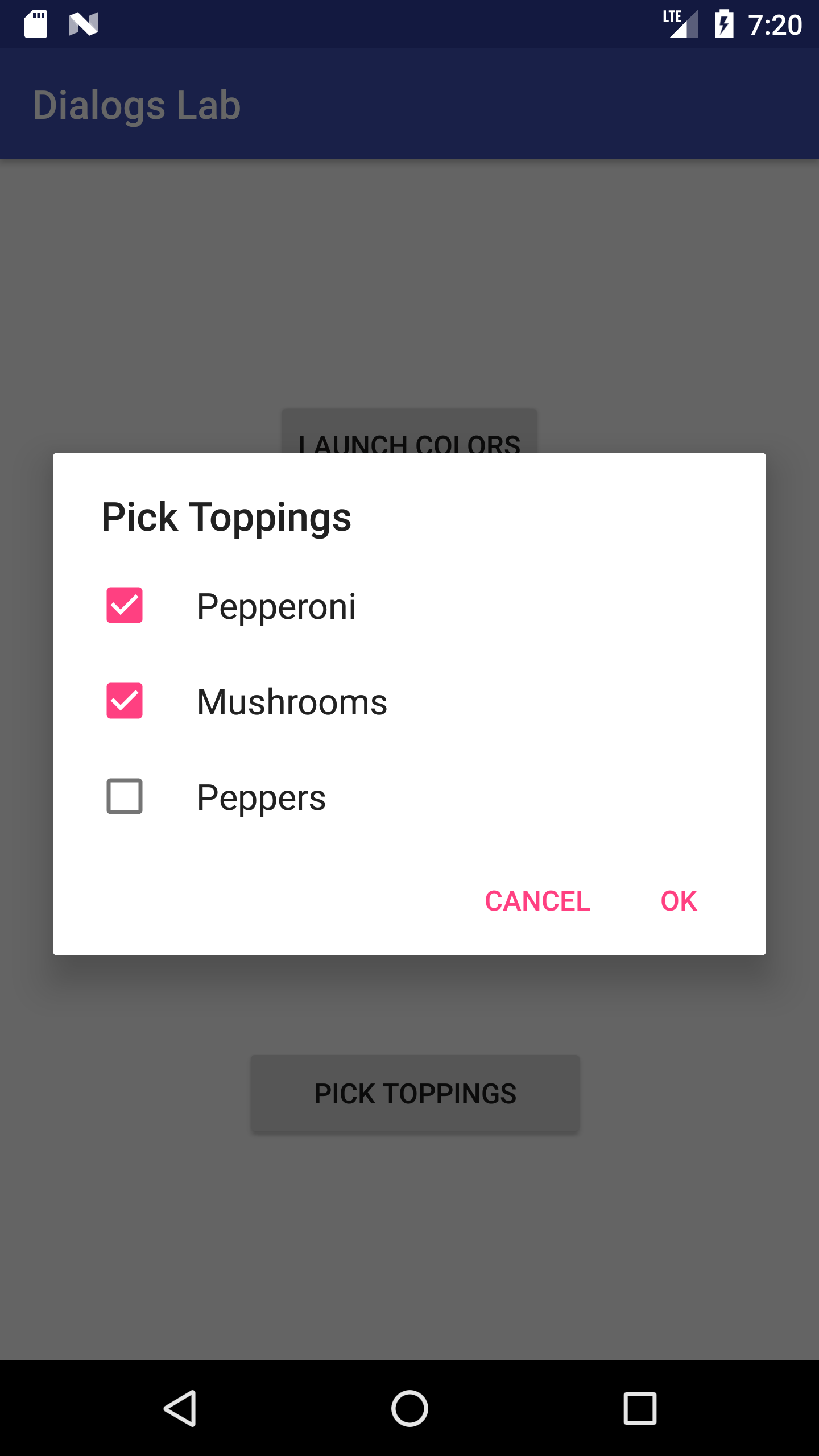
**if (fragment != null)**

**fragment.show(getSupportFragmentManager(), "launch");**

**}**

1. Run the app to test.

**Dialogs – Multi-List**



1. Create a new ***MultiListDialogFragment*** similar to the previous fragments.
2. Add the new strings to show toppings into ***strings.xml***.

**<string name="pick\_toppings">Pick Toppings</string>  
<array name="toppings\_array">  
 <item name="pepperoni">Pepperoni</item>  
 <item name="mushrooms">Mushrooms</item>  
 <item name="peppers">Peppers</item>  
</array>  
<string name="ok">OK</string>**

**<string name="launch\_multi\_dialog">Launch Multiple Dialog</string>**

1. Override the ***onCreateDialog*** method to show the multi-list.

**@Override  
public Dialog onCreateDialog(Bundle savedInstanceState) {  
 final List mSelectedItems = new ArrayList(); // Where we track the selected items  
 AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
 // Set the dialog title  
 builder.setTitle(R.string.pick\_toppings)  
 // Specify the list array, the items to be selected by default (null for none),  
 // and the listener through which to receive callbacks when items are selected  
 .setMultiChoiceItems(R.array.toppings\_array, null,  
 new DialogInterface.OnMultiChoiceClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which,  
 boolean isChecked) {  
 if (isChecked) {  
 // If the user checked the item, add it to the selected items  
 mSelectedItems.add(which);  
 } else if (mSelectedItems.contains(which)) {  
 // Else, if the item is already in the array, remove it  
 mSelectedItems.remove(Integer.valueOf(which));  
 }  
 }  
 })  
 // Set the action buttons  
 .setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int id) {  
 // User clicked OK, so save the mSelectedItems results somewhere  
 // or return them to the component that opened the dialog  
 Resources res = getActivity().getResources();  
 String[] toppings = res.getStringArray(R.array.toppings\_array);  
 StringBuilder builder = new StringBuilder();  
 for (int i=0; i<mSelectedItems.size(); i++) {  
 builder.append(toppings[(int) mSelectedItems.get(i)]);  
 builder.append(" ");  
 }  
  
 Toast.makeText(getActivity(), builder.toString() , Toast.LENGTH\_LONG)  
 .show();  
 }  
 })  
 .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int id) {  
  
 }  
 });  
  
 return builder.create();  
}**

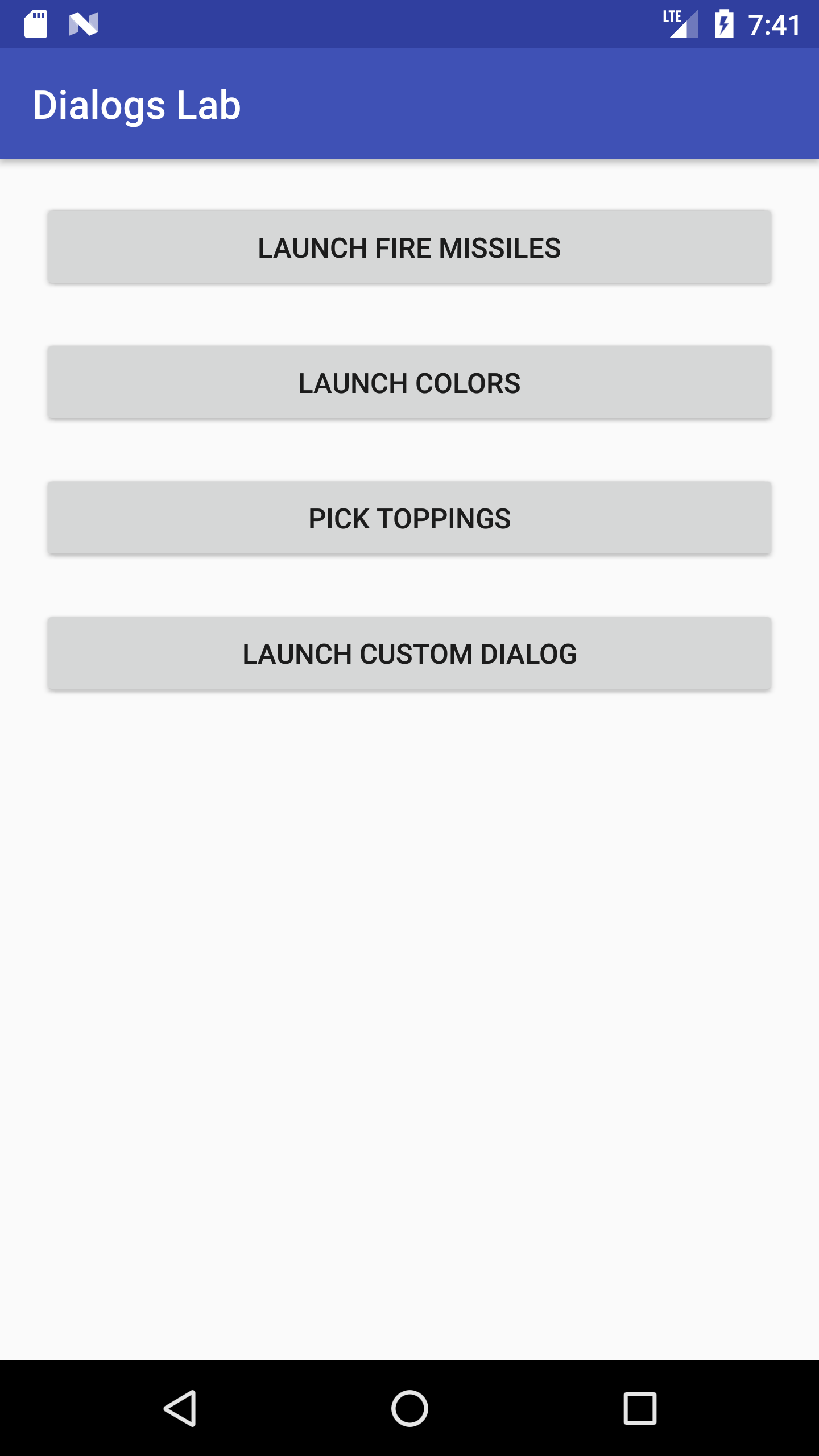
1. Add a button to launch the fragment in *MainActivity’s* xml.  
     
   **android:id="@+id/btn\_launch\_toppings"**

**android:text="@string/pick\_toppings"**

**android:onClick="launch"**

1. Modify the launch method to launch the MultiListDialogFragment.  
     
   **public void** launch(View view) {  
     
    DialogFragment fragment = **null**;  
    **if** (view.getId() == R.id.***btn\_fire\_missiles***) {  
    fragment = **new** FireMissilesDialogFragment();  
    } **else if** (view.getId() == R.id.***btn\_launch\_colors***) {  
    fragment = **new** ListDialogFragment();  
    } **else if (view.getId() == R.id.*btn\_launch\_toppings*) {  
    fragment = new MultiListDialogFragment();  
    }**  
     
     
    **if** (fragment != **null**)  
    fragment.show(getSupportFragmentManager(), **"launch"**);  
   }
2. Run the app.

**Dialogs – Custom**

****

1. Create a new ***CustomDialogFragment*** similar to the previous fragments except with a ***layout*** file.
2. Add the new strings to ***strings.xml***.

**<string name="username">Enter username</string>  
<string name="password">Enter password</string>  
<string name="signin">Sign In</string>  
<string name="launch\_custom\_dialog">Launch Custom Dialog</string>**

1. Change the layout of resource file for ***CustomDialogFragment*** to use LinearLayout as the root with two EditText views in it for username and password.

**<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="edu.uw.tacoma.mmuppa.dialogslab.CustomDialogFragment"  
 android:orientation="vertical">  
  
  
 <EditText  
 android:id="@+id/edit\_person\_name"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:inputType="textPersonName"  
 android:hint="@string/username"/>  
  
 <EditText  
 android:id="@+id/edit\_password"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="@string/password"  
 android:inputType="textPassword"/>  
  
</LinearLayout>**

1. Override the ***onCreateDialog*** method to show the custom layout.

**@Override  
public Dialog onCreateDialog(Bundle savedInstanceState) {  
 AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
 // Get the layout inflater  
 LayoutInflater inflater = getActivity().getLayoutInflater();  
  
 // Inflate and set the layout for the dialog  
 // Pass null as the parent view because its going in the dialog layout  
 builder.setView(inflater.inflate(R.layout.fragment\_custom\_dialog, null))  
 // Add action buttons  
 .setPositiveButton(R.string.signin, new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int id) {  
 // sign in the user ...  
 }  
 })  
 .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 dismiss();  
 }  
 });  
 return builder.create();  
}**

1. Add a button to launch the fragment in ***MainActivity’s*** xml.
2. Modify the launch method to launch the CustomDialogFragment.  
     
   **public void launch(View view) {  
     
    DialogFragment fragment = null;  
    if (view.getId() == R.id.btn\_fire\_missiles) {  
    fragment = new FireMissilesDialogFragment();  
    } else if (view.getId() == R.id.btn\_launch\_colors) {  
    fragment = new ListDialogFragment();  
    } else if (view.getId() == R.id.btn\_launch\_toppings) {  
    fragment = new MultiListDialogFragment();  
    }else if (view.getId() == R.id.btn\_custom\_signin) {  
    fragment = new CustomDialogFragment();  
    }  
     
     
     
     
    if (fragment != null)  
    fragment.show(getSupportFragmentManager(), "launch");  
   }**
3. Run the app.
4. (On your own) Create a notification to show up on the device. You may add another button to show the notification. The notification must use an icon (not the default Android icon) and title must contain a custom title and custom content. *Hint: You can access other Android builtin icons by using android.R.drawable.*