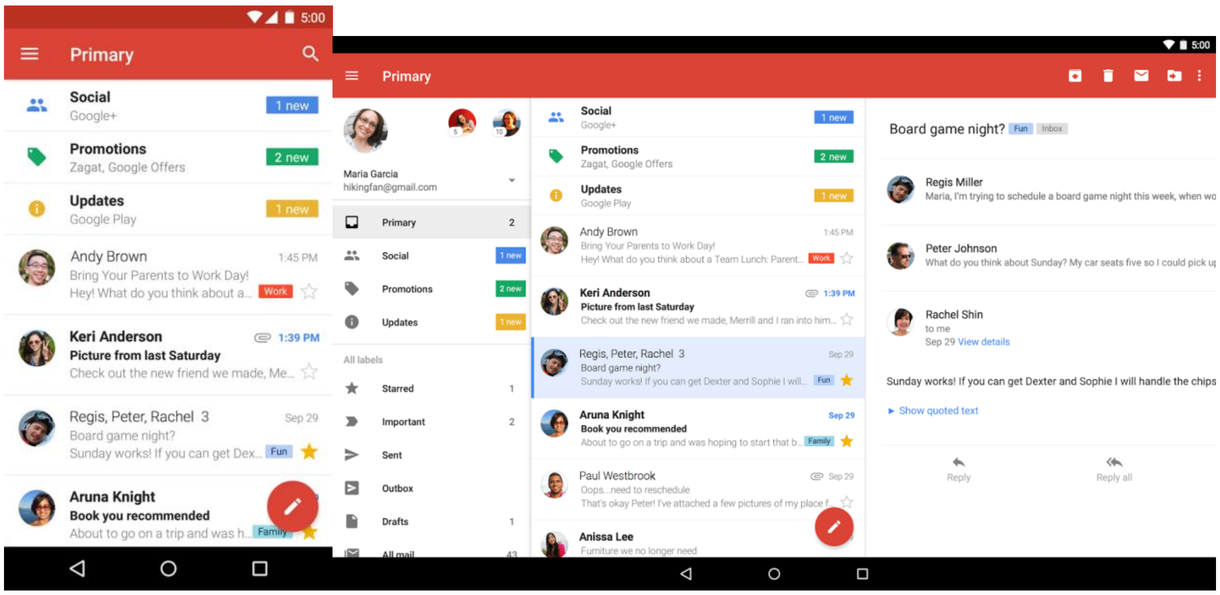
# Android Fragments Notes + Example

*IMPORTANT: **You are not required to use Fragments for your assignments or project**. They are sometimes more trouble than they are worth, but they are worth learning about and a good practice in Android because they allow for reuse of UI components. Also, some useful UI components like DatePicker and TimePicker in Android are implemented using Fragments.*

## **What do we do in Android if we want UI components that are reusable**?

For example, take a look at the Android Gmail app. When you're using an Android phone, the first screen that shows up is the inbox (picture on the left). When you're on a tablet, we see the exact same inbox interface in the middle but with other additional UI components. **However, if we were to implement the inbox UI using an Activity, we would not be able to reuse it for the tablet layout, because an Activity represents an entire screen.**



Andy Brown 1:45 PM
Bring Your Parents to Work Day!
Hey! What do you think about a Team Lunch: Parent... Work

Keri Anderson 1:39 PM
Picture from last Saturday
Check out the new friend we made, Merrill and I ran into him...

Regis, Peter, Rachel 3 Sep 29
Board game night?
Sunday works! If you can get Dexter and Sophie I will... Fun

Aruna Knight Sep 29
Book you recommended
About to go on a trip and was hoping to start that b... Family

Paul Westbrook Sep 29
Oops...need to reschedule
That's okay Peter! I've attached a few pictures of my place f...

Anissa Lee
Furniture we no longer need

Board game night? Fun Inbox

Regis Miller
Maria, I'm trying to schedule a board game night this week, when wo

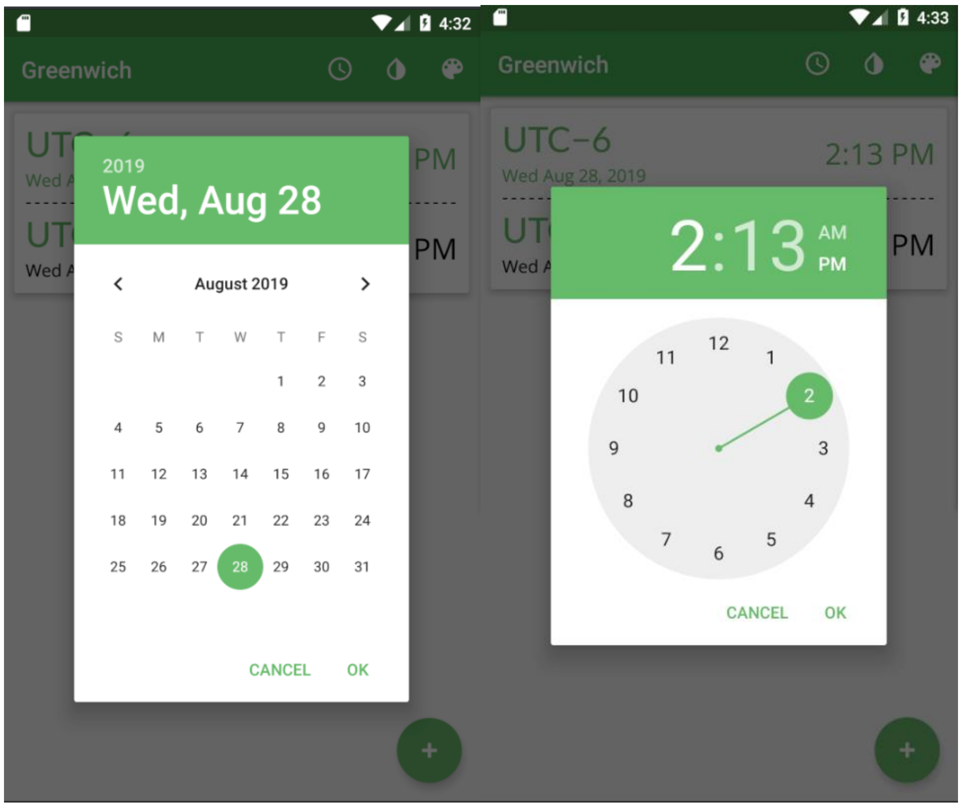
Peter Johnson
What do you think about Sunday? My car seats five so I could pick up

Rachel Shin
to me
Sep 29 View details

Sunday works! If you can get Dexter and Sophie I will handle the chips

This is where Fragments come in. You can think of them as "mini Activities", except the difference is that **you can define a Fragment once and use it in multiple Activities. Fragments can only be shown when it is hosted by an Activity, and you can also display multiple Fragments in the same Activity.**

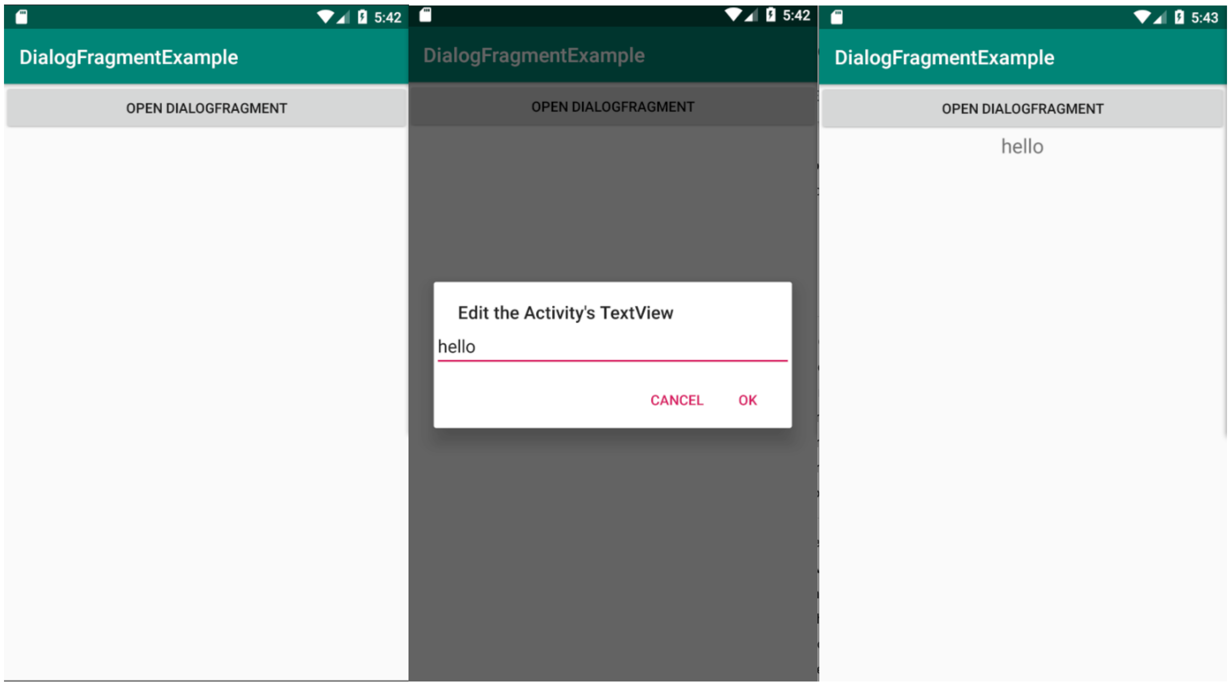
Two examples of Fragments are Android's built in **DatePicker** and **TimePicker** shown below. They allow you to choose a date and time using a fancy looking UI.



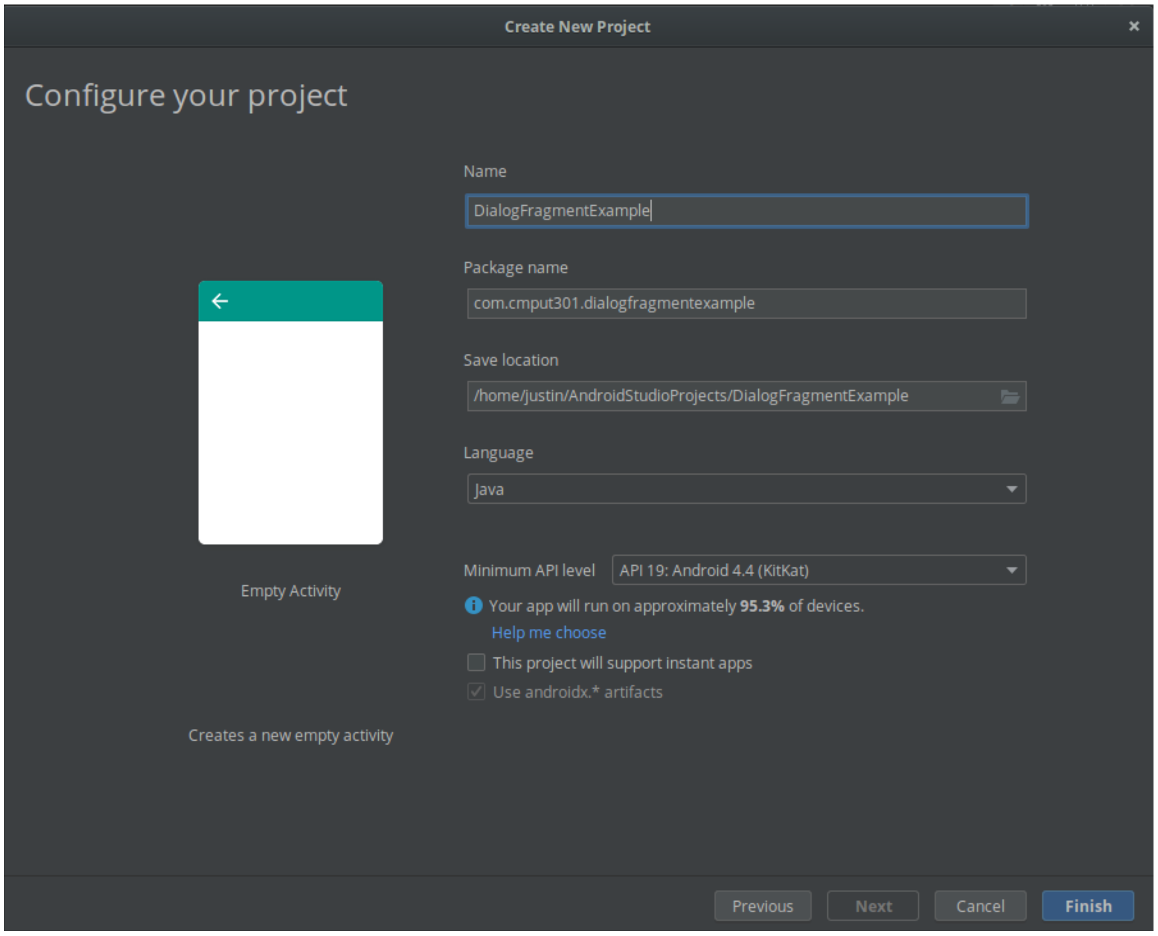
**Why did the Android developers decide to make these a Fragment instead of an Activity?** A major reason is that it would be poor UX design if you had to open up a new screen every time you wanted to select a date or time. Also, the DatePicker and TimePicker widgets do not even fill the whole screen, so they do not require an Activity on their own. To use these in your apps, you simply need to start up the Fragment from an Activity send the selected date or time into the Activity.

## EXERCISE

**To demonstrate usage of Fragments and how to send data from a Fragment back to its hosting Activity, we'll make an app that has a single Activity with a Button and a TextView. When you press the Button, the Activity opens up a DialogFragment which has an EditText. When you type something into the EditText and press OK, the text will be shown in the Activity's TextView.**



**Start a new empty Android project. Make sure the Language selected is "Java"**



Create New Project

Configure your project

Name: DialogFragmentExample

Package name: com.cmput301.dialogfragmentexample

Save location: /home/justin/AndroidStudioProjects/DialogFragmentExample

Language: Java

Minimum API level: API 19: Android 4.4 (KitKat)

Your app will run on approximately **95.3%** of devices.

Help me choose

☐ This project will support instant apps

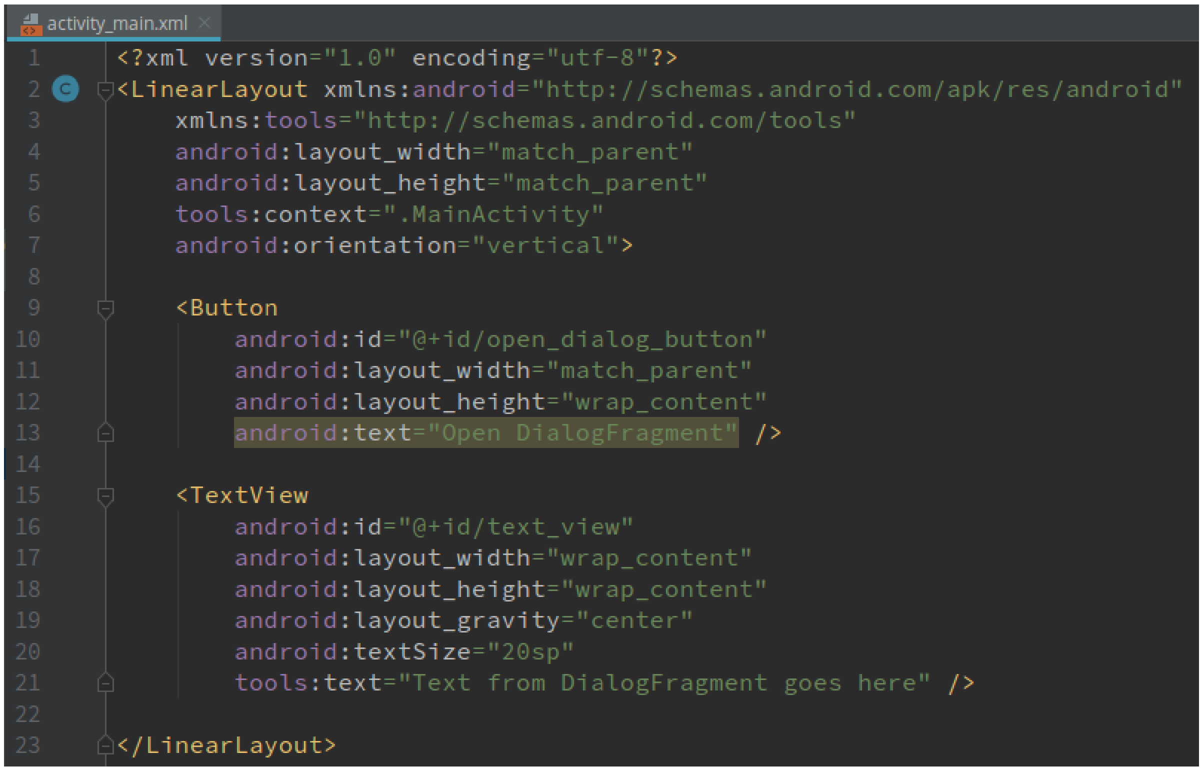
☑ Use androidx.* artifacts

Empty Activity

Creates a new empty activity

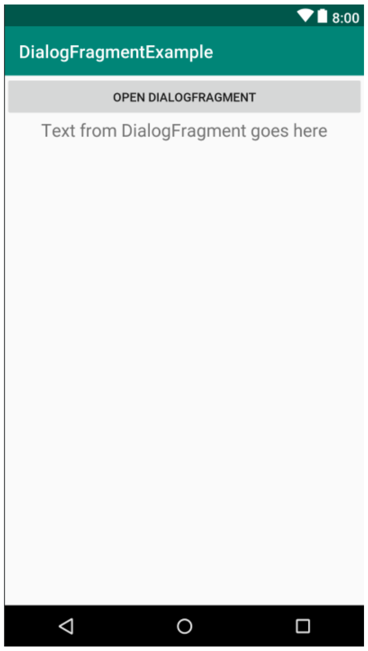
**First, let’s update the MainActivity’s layout file to use a LinearLayout for its ViewGroup, and also add a Button and a TextView.**

**activity_main.xml**

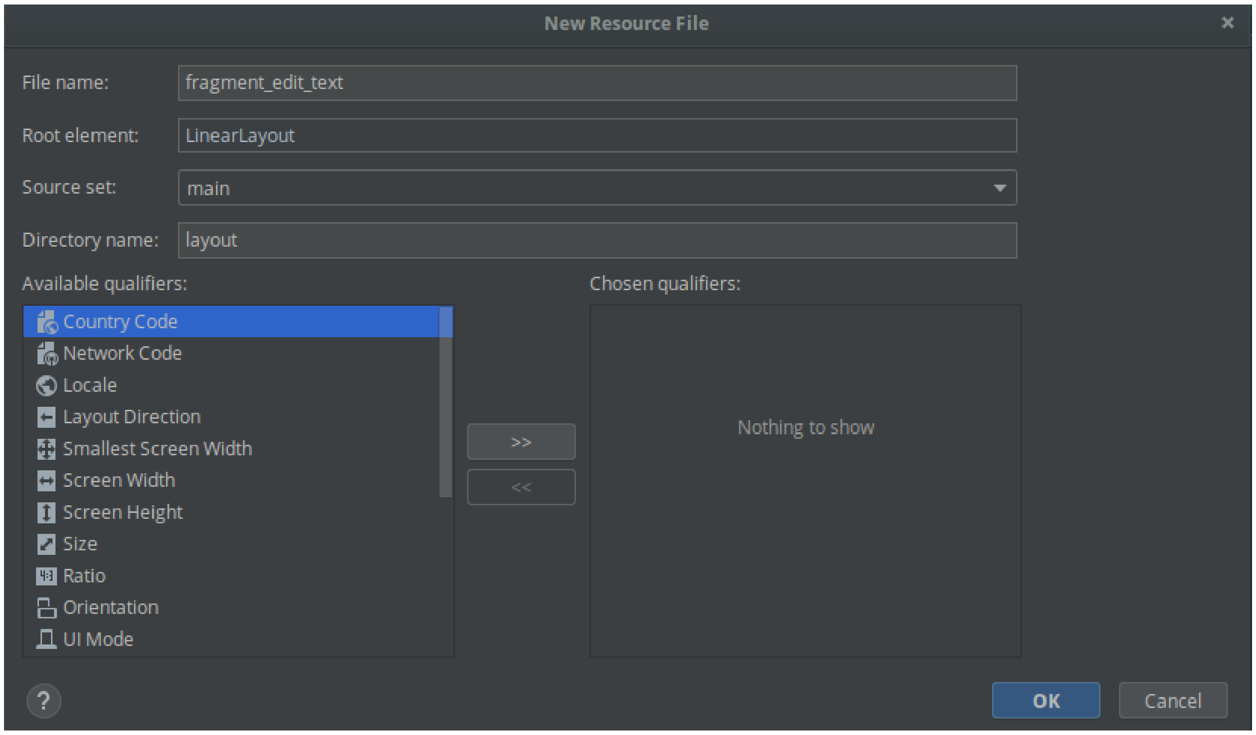


activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<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=".MainActivity"
    android:orientation="vertical">

    <Button
        android:id="@+id/open_dialog_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Open DialogFragment" />

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:textSize="20sp"
        tools:text="Text from DialogFragment goes here" />

</LinearLayout>
```



**Make a new layout file called fragment_edit_text. This will be the layout that our Fragment uses.**



File name: fragment_edit_text

Root element: LinearLayout

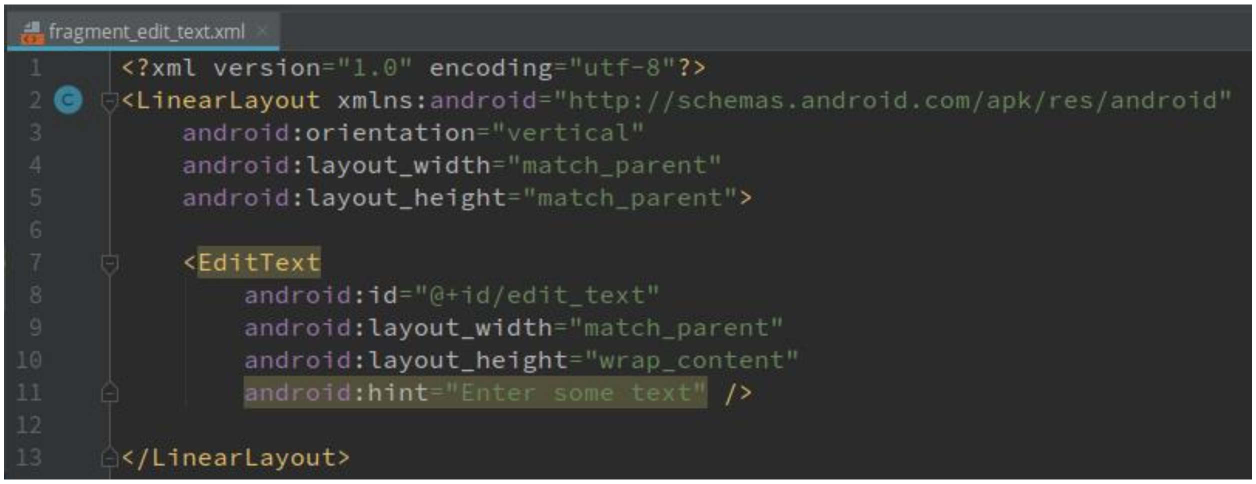
Source set: main

Directory name: layout

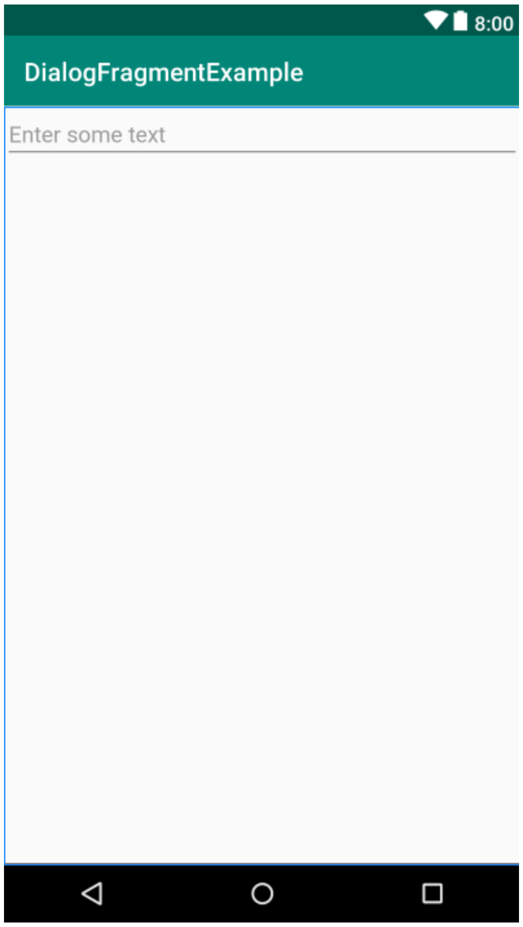
Available qualifiers:

Chosen qualifiers:

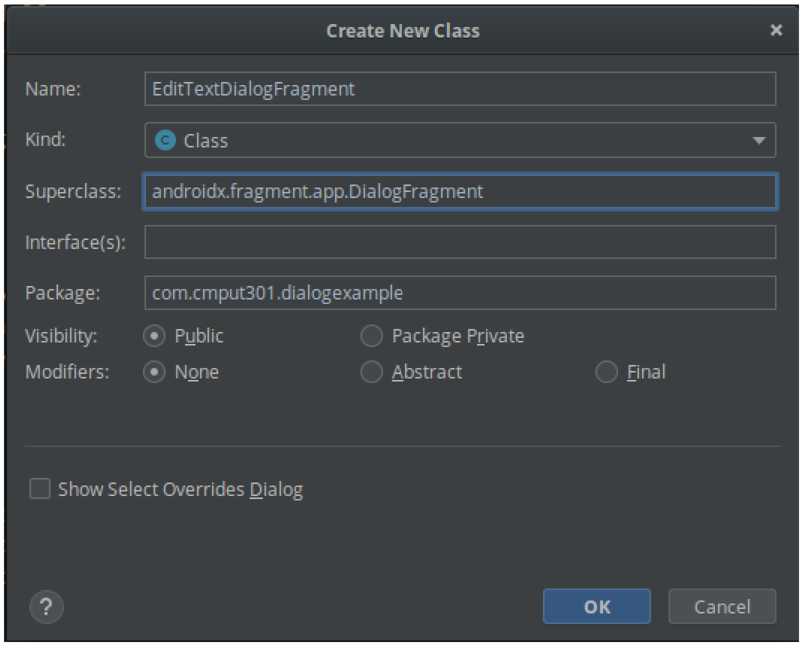
**Add an EditText to the fragment_edit_text.xml**.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter some text" />

</LinearLayout>
```



**Next, let's make the Java class for the DialogFragment. Name it EditTextDialogFragment and make it extend DialogFragment.**



Name: EditTextDialogFragment

Kind: Class

Superclass: androidx.fragment.app.DialogFragment

Interface(s):

Package: com.cmput301.dialogexample

Visibility: [✓ Public] Package Private

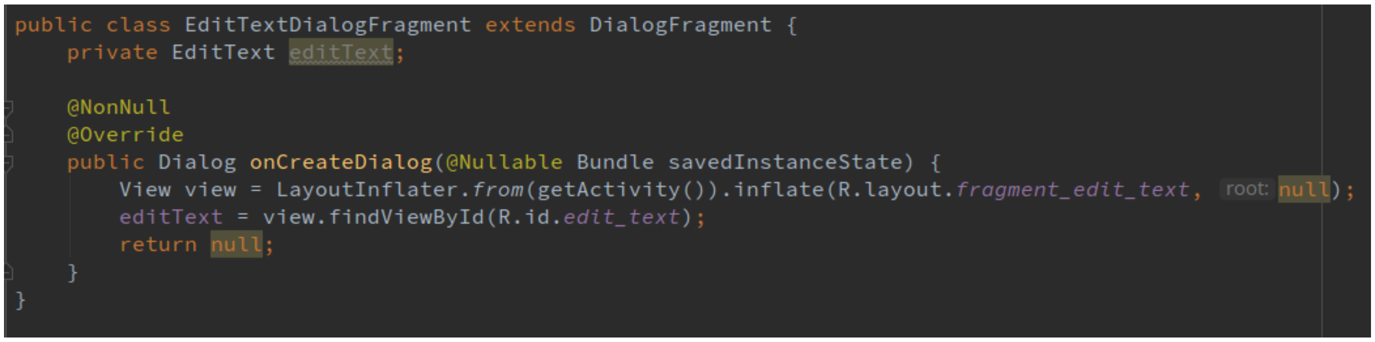
Modifiers: [✓ None] Abstract Final

☐ Show Select Overrides Dialog

Since we’re using a DialogFragment, we want to **override the “onCreateDialog” method.**

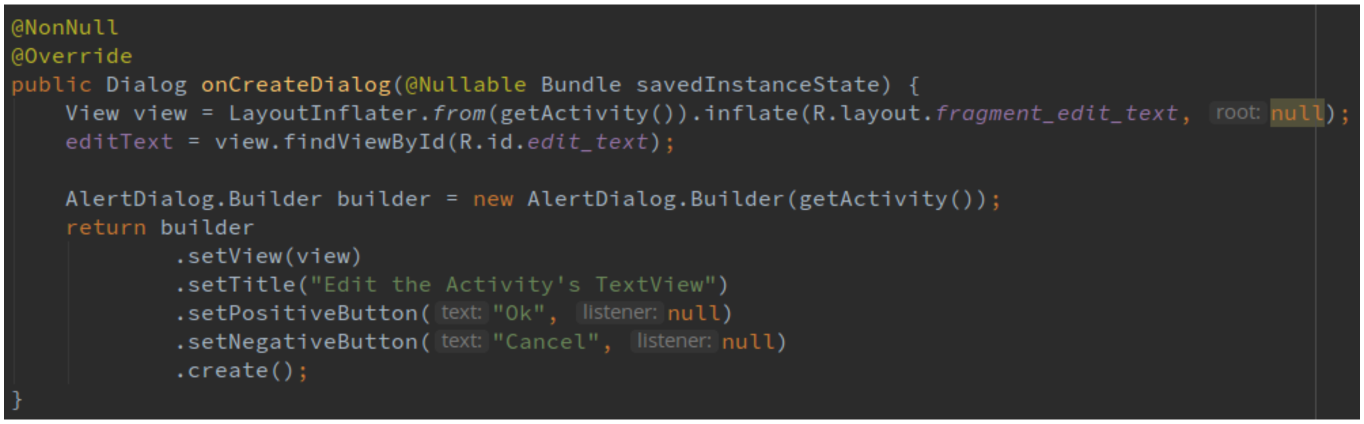
From this method, we return a Dialog which is what we want the Activity to display when the button is clicked. First, we need to inflate the Fragment layout file we just made. “Inflating” the layout file gives us the actual View object. **In this case, the View that is returned from .inflate() will be the root View of fragment_edit_text.xml, which is the LinearLayout.**

Also, get a reference to the EditText. This will allow us to pull the text out of it later and send it to the Activity.



```
public class EditTextDialogFragment extends DialogFragment {
    private EditText editText;

    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        View view = LayoutInflater.from(getActivity()).inflate(R.layout.fragment_edit_text,  root: null);
        editText = view.findViewById(R.id.edit_text);
        return null;
    }
}
```

**Next, we use a class called AlertDialog.Builder to build the actual Dialog**. First we make a builder, then call specific methods on the builder to edit parts of the Dialog. Last, make sure to call .create() on the Builder to actually get an instance of the Dialog. We return this newly created Dialog object.

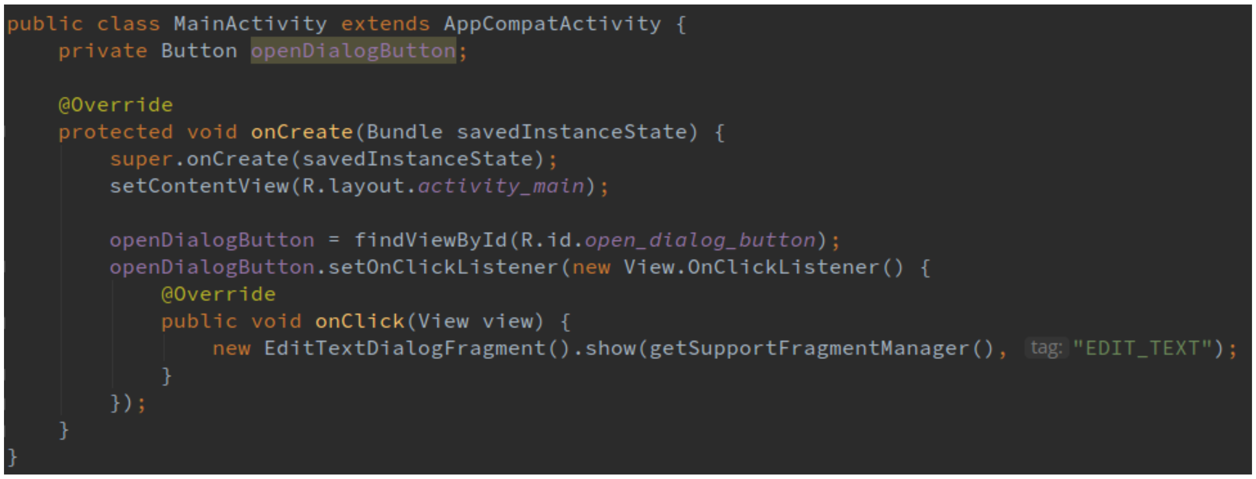


```
@NonNull
@Override
public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
    View view = LayoutInflater.from(getActivity()).inflate(R.layout.fragment_edit_text,  root: null);
    editText = view.findViewById(R.id.edit_text);

    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    return builder
            .setView(view)
            .setTitle("Edit the Activity's TextView")
            .setPositiveButton( text: "Ok",  listener: null)
            .setNegativeButton( text: "Cancel",  listener: null)
            .create();
}
```

**In MainActivity, add a View.OnClickListener to the Button, and here we will open up the DialogFragment when it is clicked.**

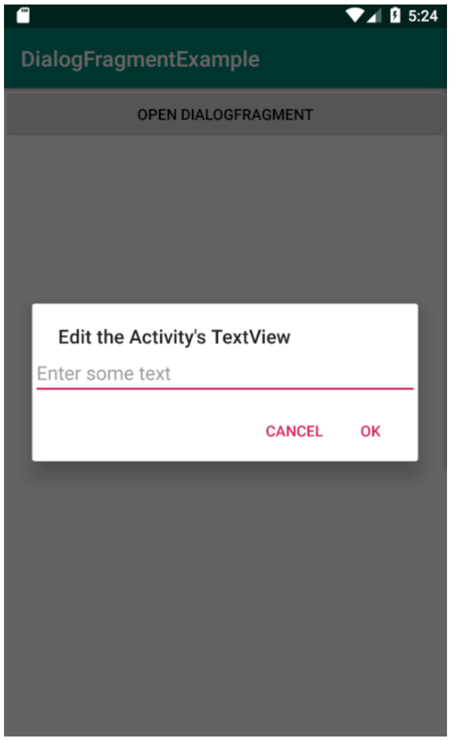
**The way that an Activity opens up a Fragment is using a “FragmentManager”**. Usually, the Activity must specify what Fragment it wants opened up, and also a tag (just a String) for the FragmentManager to identify the Fragment later, since an Activity can have multiple Fragments.

**However, the code is a bit different with DialogFragments.** First, we make a new instance of the EditTextDialogFragment class we just created, and we call .show() on it. We provide the FragmentManager and tag to the show method.



```
public class MainActivity extends AppCompatActivity {
    private Button openDialogButton;

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

        openDialogButton = findViewById(R.id.open_dialog_button);
        openDialogButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                new EditTextDialogFragment().show(getSupportFragmentManager(), tag: "EDIT_TEXT");
            }
        });
    }
}
```

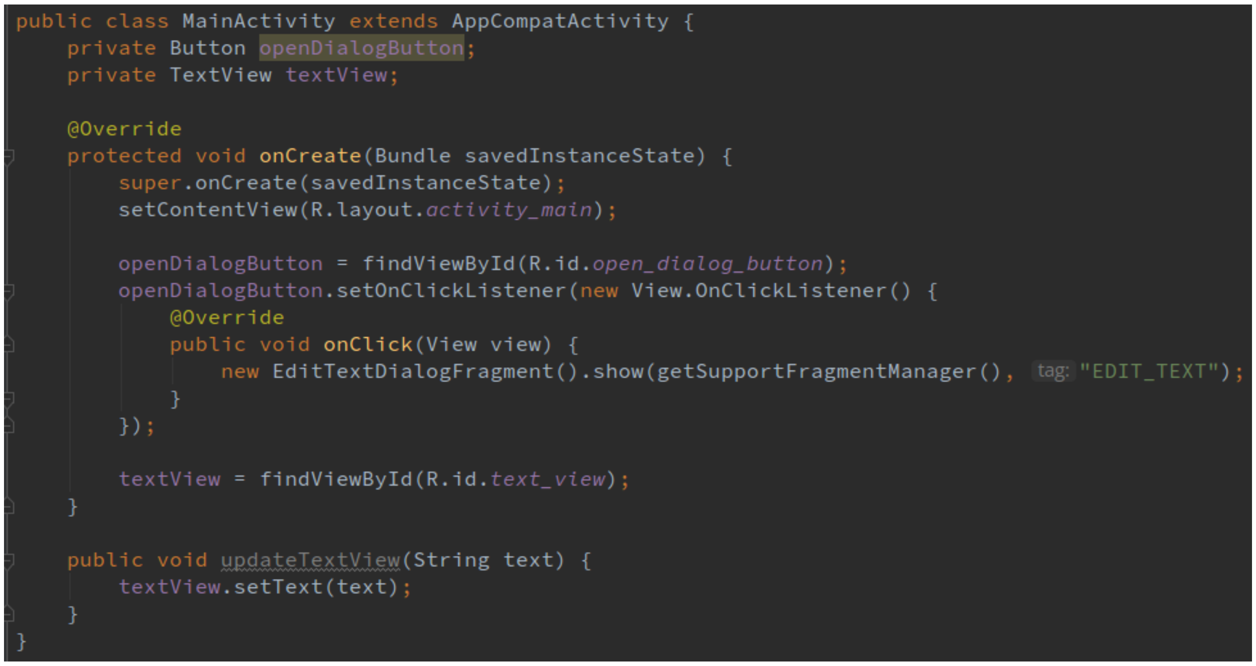
**Run the app and ensure that the dialog opens when the button is pressed!**



It's looking great. However, the EditText doesn't do anything even after pressing OK. **We want to update the TextView in MainActivity with whatever the user enters in the DialogFragment's EditText.**

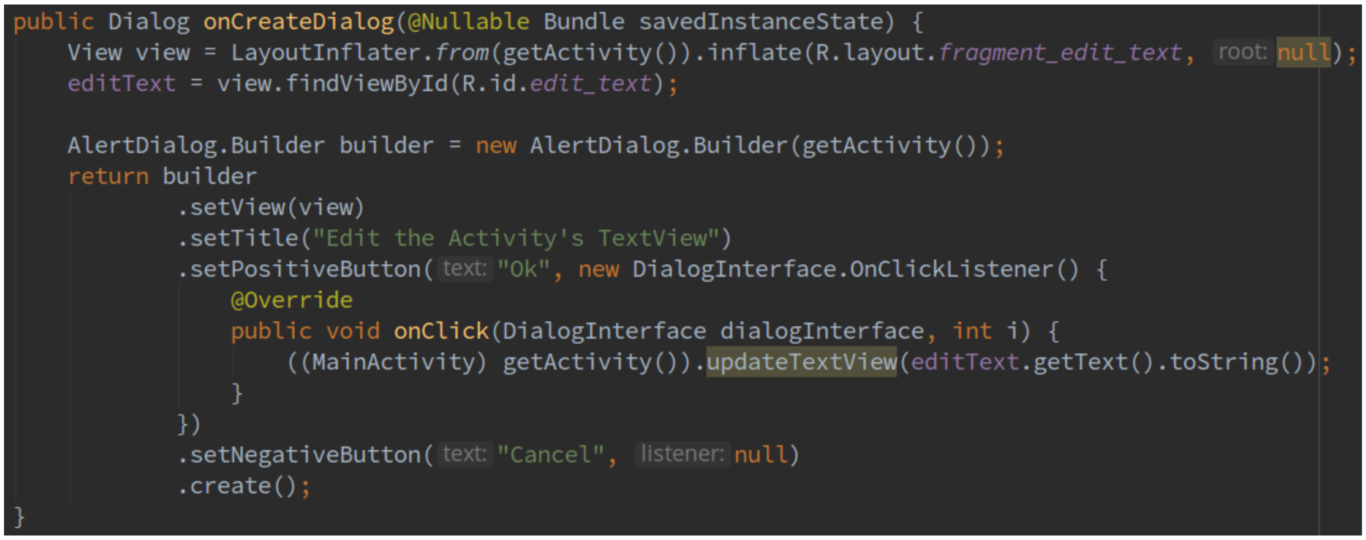
**One way we can send the text from the Fragment to the Activity is the following:**

Get a reference to the TextView in MainActivity, provide a method updateTextView that takes in a String, and sets the String onto the TextView.



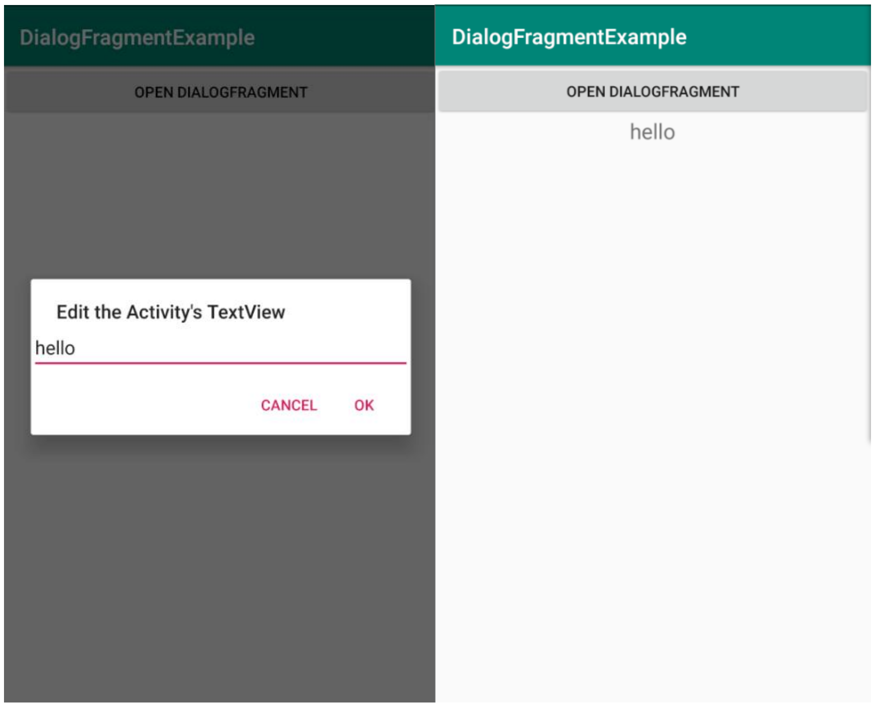
```java
public class MainActivity extends AppCompatActivity {
    private Button openDialogButton;
    private TextView textView;

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

        openDialogButton = findViewById(R.id.open_dialog_button);
        openDialogButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                new EditTextDialogFragment().show(getSupportFragmentManager(), tag: "EDIT_TEXT");
            }
        });

        textView = findViewById(R.id.text_view);
    }

    public void updateTextView(String text) {
        textView.setText(text);
    }
}
```

In EditTextDialogFragment, add a DialogInterface.onClickListener for the “OK” button. (this is extremely similar to View.OnClickListener). **We can call getActivity(), cast it to a type of MainActivity, and send in the EditText’s text.**



```java
public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
    View view = LayoutInflater.from(getActivity()).inflate(R.layout.fragment_edit_text, root: null);
    editText = view.findViewById(R.id.edit_text);

    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    return builder
            .setView(view)
            .setTitle("Edit the Activity's TextView")
            .setPositiveButton( text: "Ok", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    ((MainActivity) getActivity()).updateTextView(editText.getText().toString());
                }
            })
            .setNegativeButton( text: "Cancel", listener: null)
            .create();
}
```

**Run your app and open the DialogFragment. Input some text and press OK - the Activity's TextView should now be updated!**



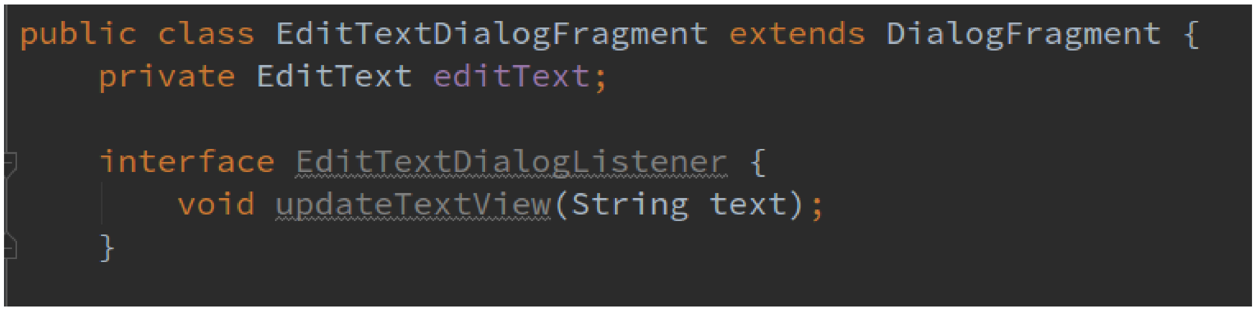
**This works, but unfortunately if we implement it this way, it destroys the whole "reusability" of Fragments.**

```
@Override
public void onClick(DialogInterface dialogInterface, int i) {
    ((MainActivity) getActivity()).updateTextView(editText.getText().toString());
}
```

**As we can see above, the Fragment depends on its hosting Activity being of type MainActivity. What if we wanted to host this DialogFragment in another Activity, that isn't of type MainActivity? It's now extremely difficult to do that.**

We can see that this would be an issue if Android implemented their DatePicker or TimePicker fragments in this way. No one would be able to use them and you would be forced to start up another Activity just to select a date or time.

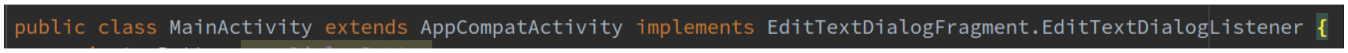
**We fix this by using an interface. First, define an interface on the Fragment.**



```java
public class EditTextDialogFragment extends DialogFragment {
    private EditText editText;

    interface EditTextDialogListener {
        void updateTextView(String text);
    }
```

**Now, any Activity that wants to host this Fragment must implement this interface.**

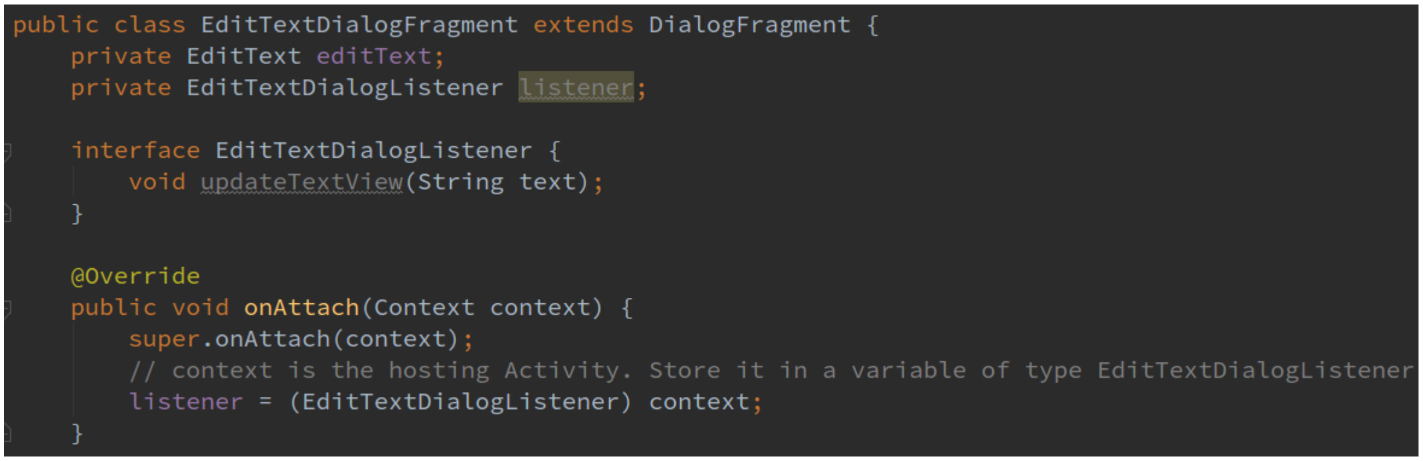
**Update your MainActivity to implement the interface.**



```java
public class MainActivity extends AppCompatActivity implements EditTextDialogFragment.EditTextDialogListener {
```

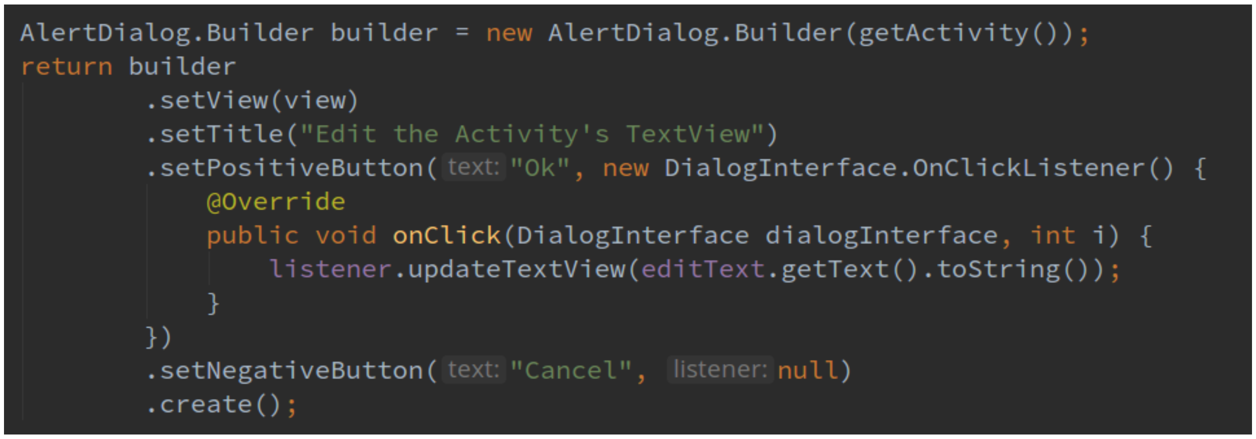
Since our MainActivity already has the updateTextView method, the Activity doesn't violate the interface and Android Studio shouldn't complain.

Now, the key part is that there is a **Fragment lifecycle method called "onAttach". This method gets called when the Fragment is being attached to its hosting Activity. The method takes in a Context, which is the hosting Activity** (in our case, it's MainActivity). Here is where we get a reference to the hosting activity and store it in a type of EditTextDialogListener. This works because MainActivity implements EditTextDialogListener, so we can refer to it using a variable of that type.



```java
public class EditTextDialogFragment extends DialogFragment {
    private EditText editText;
    private EditTextDialogListener listener;

    interface EditTextDialogListener {
        void updateTextView(String text);
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        // context is the hosting Activity. Store it in a variable of type EditTextDialogListener
        listener = (EditTextDialogListener) context;
    }
```

**Now, in the onClick method of the Ok button, we call the updateTextView method on the interface instead.**



```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
return builder
        .setView(view)
        .setTitle("Edit the Activity's TextView")
        .setPositiveButton( text: "Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                listener.updateTextView(editText.getText().toString());
            }
        })
        .setNegativeButton( text: "Cancel",  listener: null)
        .create();
```

**Notice how we never reference MainActivity specifically. Our Fragment is now reusable and can be hosted/used by different Activities!** Rerun the app and it should work exactly like it did before.

It's interesting to note that our implementation above to send data from the Fragment to the Activity using the interface above is actually how Android's DatePicker and TimePicker DialogFragments are implemented. When you want to get the date/time back from them, you must implement their OnDateSetListener or OnTimeSetListener to get the result back.