



Android Dialogs

In this Lecture, you will learn:

- Creating Dialog
-

Dialog

A dialog is a window that prompts the user to make a decision or enter additional information. A dialog is normally used for modal events that require users to take an action before they can proceed.

The Dialog class is the base class for dialogs. Use one of the following subclasses:

AlertDialog

A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.

DatePickerDialog or TimePickerDialog are pre-defined UI that allows the user to select a date or time.

The DialogFragment class provides all the controls you need to create your dialog and manage its appearance.

```
AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
builder.setMessage("Are you sure you want to move to Mars?")
    .setPositiveButton("YES", new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        Toast.makeText(MainActivity.this, "Fired!",
Toast.LENGTH_SHORT).show();
    }
})
    .setNegativeButton("NO", new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        Toast.makeText(MainActivity.this, "Stay safe",
Toast.LENGTH_SHORT).show();
    }
});
builder.create().show();
```

Building an Alert Dialog:

The `AlertDialog` class allows you to build a variety of dialog designs and is often the only dialog class you'll need. As shown in figure, there are three regions of an alert dialog:

Title

This optional field should be used when the content area is occupied by a detailed message, a list, or custom layout.

Content area

This can display a message, a list, or other custom layout.

Action buttons

There should be no more than three action buttons in a dialog.

The `AlertDialog.Builder` class provides APIs that allow you to create an `AlertDialog` with these kinds of content, including a custom layout.

To build an `AlertDialog`:

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setMessage(R.string.dialog_message)
    .setTitle(R.string.dialog_title);
AlertDialog dialog = builder.create();
```

Adding buttons

To add action buttons, call the `setPositiveButton()` and `setNegativeButton()` methods:

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setPositiveButton(R.string.ok, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
    }
});
builder.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
```

```
    }  
});  
AlertDialog dialog = builder.create();
```

The set...Button() methods require a title for the button (supplied by a string resource) and a DialogInterface.OnClickListener that defines the action to take when the user presses the button.

There are three different action buttons you can add:

Positive

You should use this to accept and continue with the action (the "OK" action).

Negative

You should use this to cancel the action.

Neutral

You should use this when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."

You can add only one of each button type to an AlertDialog. That is, you cannot have more than one "positive" button.

Adding a list

There are three kinds of lists with the AlertDialog APIs:

A single-choice list

A persistent single-choice list (radio buttons)

A persistent multiple-choice list (checkboxes)

To create a single-choice list, use the setItems() method:

```
<string name="pick_color">Pick Color</string>  
<string-array name="colors_array">  
    <item>Red</item>  
    <item>Blue</item>  
    <item>Green</item>  
</string-array>
```

```

public void singleChoiceList(){
    AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
    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
            colorsArray =
getResources().getStringArray(R.array.colors_array);
            Toast.makeText(MainActivity.this, "Item clicked "+
                colorsArray[which]
                , Toast.LENGTH_SHORT).show();
        }
    });
    builder.create().show();
}

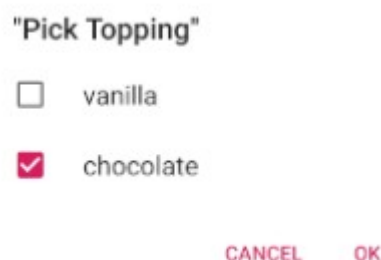
```



As the list appears in the dialog's content area, the dialog cannot show both a message and a list and you should set a title for the dialog with setTitle(). To specify items for the list, call setItems(), passing an array, or specify a list using setAdapter(). This allows you to back the list with dynamic data using a ListAdapter.

Adding a persistent multiple-choice or single-choice list:

To add a list of multiple-choice items (checkboxes) or single-choice items (radio buttons), use the setMultiChoiceItems() or setSingleChoiceItems() methods, respectively. Here's how you can create a multiple-choice list like the one shown in figure below that saves the selected items in an ArrayList:



```

<string-array name="toppings">
  <item>vanilla</item>
  <item>chocolate</item>
</string-array>

```

```

public void multiChoiceList(){
    final ArrayList selectedItems = new ArrayList(); // Where we
    track the selected items
    AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
    // 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, 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
                            selectedItems.add(which);
                        } else if
(selectedItems.contains(which)) {
                            // Else, if the item is already in
the array, remove it
                            selectedItems.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 selectedItems
results somewhere

```

```

// or return them to the component that opened
the dialog
        }
    })
    .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int
id) {
            }
        });
    builder.create().show();
}

```

Creating a Custom Layout:

If you want a custom layout in a dialog, create a layout and add it to an AlertDialog by calling `setView()` on your `AlertDialog.Builder` object.

By default, the custom layout fills the dialog window, but you can still use `AlertDialog.Builder` methods to add buttons and a title.

For example, here's the layout file for the dialog in Figure:



```

public void createCustomDialog(){
    AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
    // Get the layout inflater
    LayoutInflater inflater = 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.dialog_signin, 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) {
        //LoginDialogFragment.this.getDialog().cancel();
    }
});
builder.create().show();
}

```

dialog_signin.xml:

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/editTextUsername"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="username"
        android:inputType="textPersonName" />
    <EditText
        android:id="@+id/editTextPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPassword" />
</LinearLayout>

```

Creating a Dialog Fragment:

A basic AlertDialog managed within a DialogFragment:

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.util.Log;
public class ConfirmationDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
        builder.setMessage("Are you sure you want to move to Mars?")
            .setPositiveButton("YES", new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int
id) {
                    Log.d("OnOption", "YES Selected");
                }
            })
            .setNegativeButton("NO", new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int
id) {
                    Log.d("OnOption", "NO Selected");
                }
            });
        return builder.create();
    }
}
```

Showing a Dialog:

Create an instance of your DialogFragment and call show(), passing the FragmentManager and a tag name for the dialog fragment.

Get the FragmentManager by calling getSupportFragmentManager() from the FragmentActivity or getFragmentManager() from a Fragment. For example:

```
ConfirmationDialogFragment confirmationDialogFragment = new
ConfirmationDialogFragment();
confirmationDialogFragment.show( getSupportFragmentManager(), "showing
fragment");
```

The second argument, "missiles", is a unique tag name that the system uses to save and restore the fragment state when necessary. The tag also allows you to get a handle to the fragment by calling `findFragmentByTag()`.

Passing Events Back to the Dialog's Host

