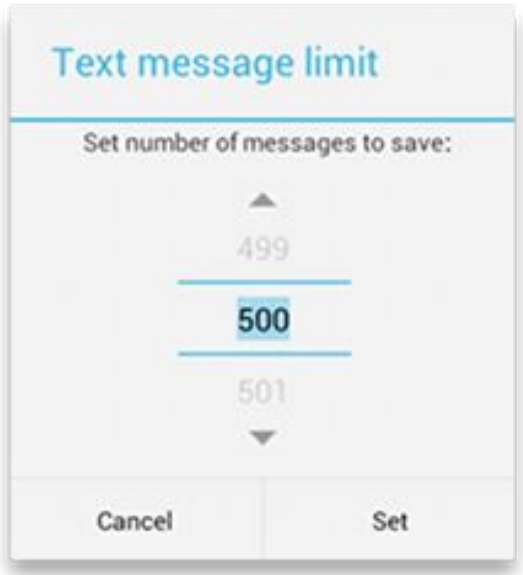
Mobile Apps Development

Topics

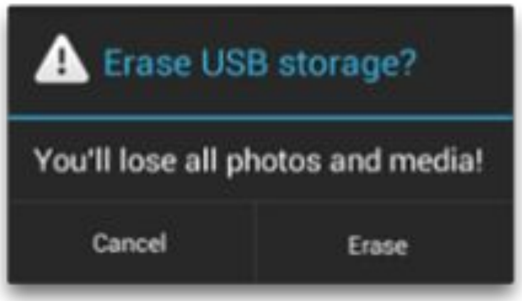
1. Dialog
2. ProgressBar & ProgressDialog
3. Dimension
4. Weight
5. Gravity
6. Sharing Information Using Intents
7. Exception Handling in Java

Dialog

- A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for specific events that require users to take an action before they can proceed.
- The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses: AlertDialog, DatePickerDialog or TimePickerDialog

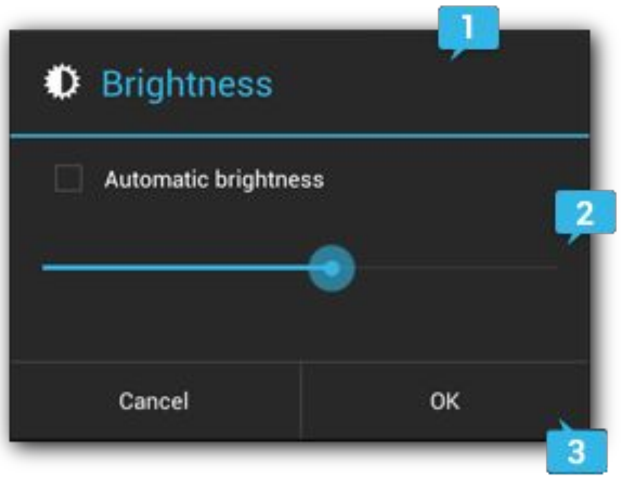

Text message limit

Set number of messages to save:


Erase USB storage?

You'll lose all photos and media!

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:

1. Title

This is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If you need to state a simple message or question, you don't need a title.

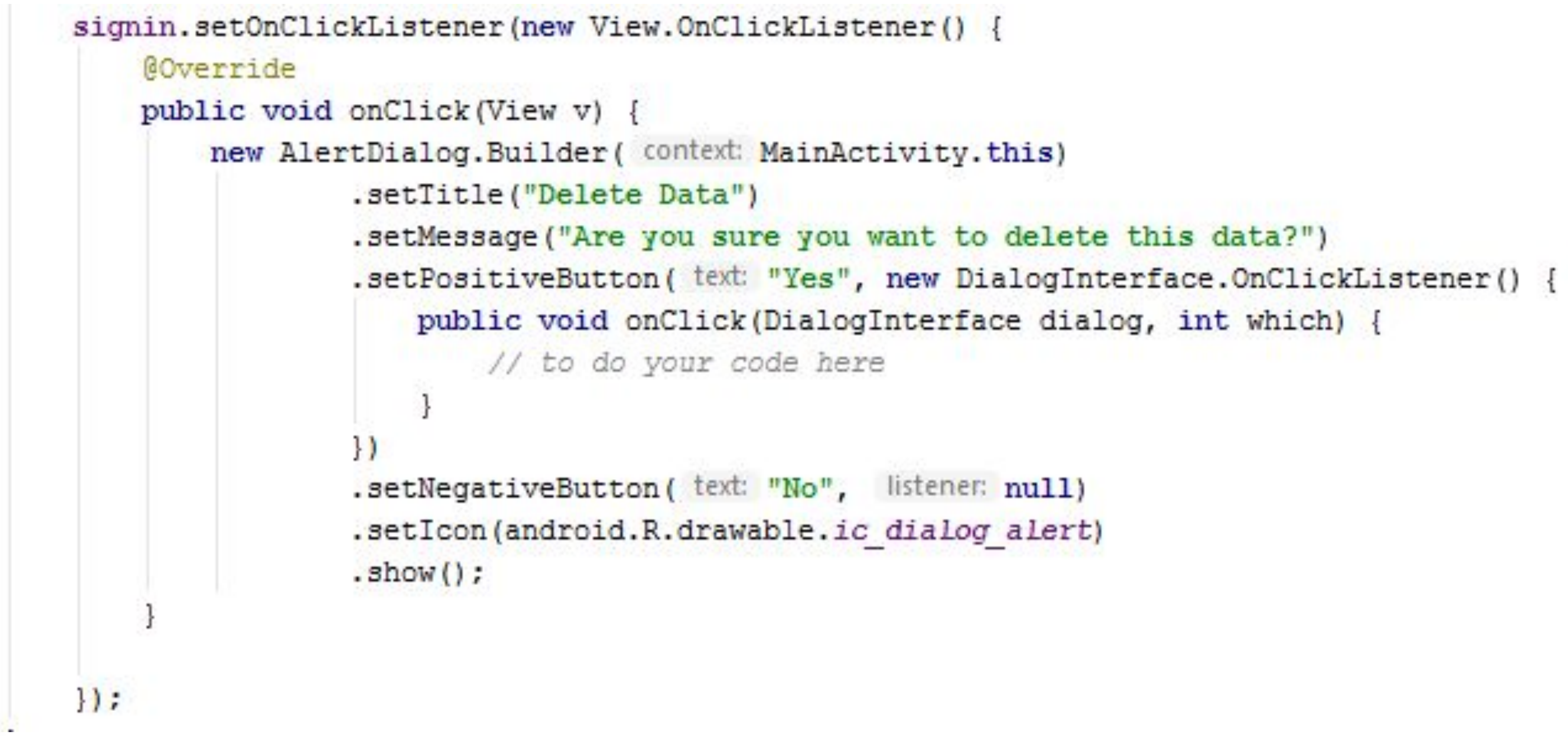
2. Content area

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

3. Action buttons

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

Building an Alert Dialog- JAVA



```
signin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new AlertDialog.Builder( context: MainActivity.this)
                .setTitle("Delete Data")
                .setMessage("Are you sure you want to delete this data?")
                .setPositiveButton( text: "Yes", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
                        // to do your code here
                    }
                })
                .setNegativeButton( text: "No",  listener: null)
                .setIcon(android.R.drawable.ic_dialog_alert)
                .show();
    }

});
```

ProgressBar

1. In Android, ProgressBar is used to display the status of work being done like analyzing status of work or downloading a file etc.
2. In Android, by default a progressbar will be displayed as a spinning wheel but If we want it to be displayed as a horizontal bar then we need to use style attribute as horizontal.
3. A user interface element that indicates the progress of an operation.
4. Progress bar supports two modes to represent progress: determinate, and indeterminate.

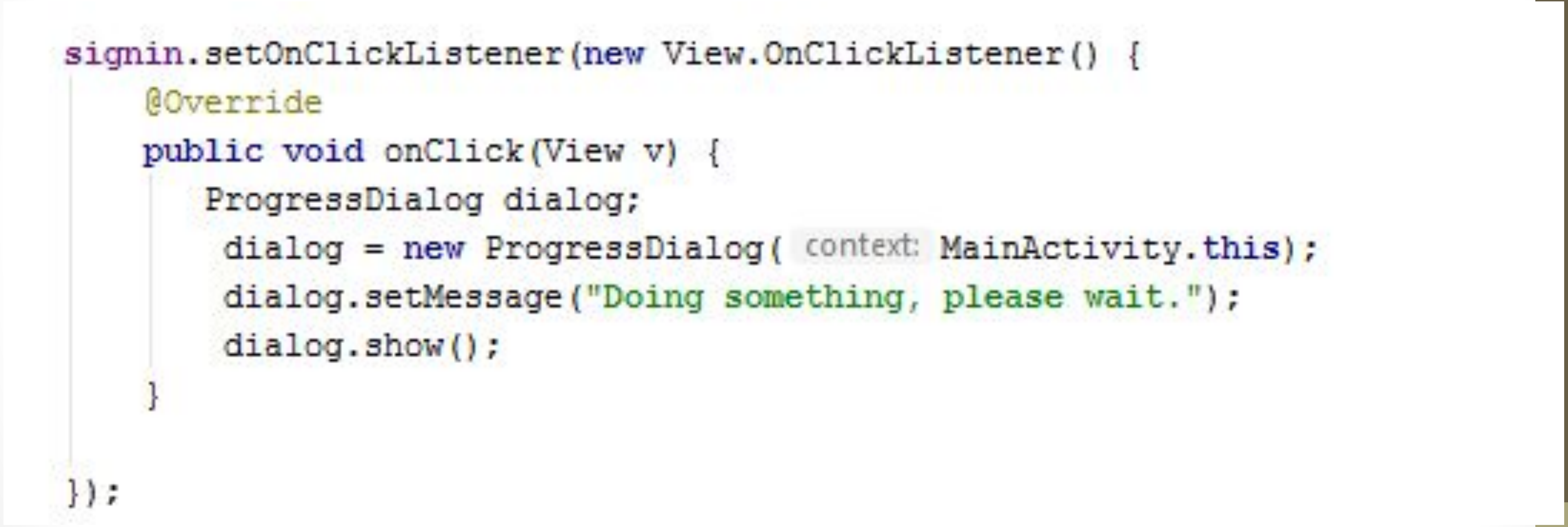
ProgressBar

1. Use indeterminate mode for the progress bar when you do not know how long an operation will take. Indeterminate mode is the default for progress bar and shows a cyclic animation without a specific amount of progress indicated.
2. Use determinate mode for the progress bar when you want to show that a specific quantity of progress has occurred. For example, the percent remaining of a file being retrieved, the amount records in a batch written to database, or the percent remaining of an audio file that is playing.

```
 <ProgressBar
android:id="@+id/indeterminateBar"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
     />
```

```
 <ProgressBar
android:id="@+id/determinateBar"
style="@android:style/Widget.ProgressBar.Horizont
al"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:progress="25"/>
```

ProgressDialog- JAVA



```
signin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ProgressDialog dialog;
         dialog = new ProgressDialog( context: MainActivity.this);
         dialog.setMessage("Doing something, please wait.");
         dialog.show();
    }

});
```

Dimensions

Dimensions are used in several places in Android to describe distances, such as a widget’s size. There are several different units of measurement available to you:

1. px means hardware pixels, whose size will vary by device, since not all devices have the same screen density
2. in and mm for inches and millimeters, respectively, based on the actual size of the screen
3. pt for points, which in publishing terms is 1/72 of an inch (again, based on the actual physical size of the screen)
4. dip (or dp) for device-independent pixels — one dip equals one hardware pixel for a ~160dpi resolution screen, but one dip equals two hardware pixels on a ~320dpi screen
5. sp for scaled pixels, where one sp equals one dip for normal font scale levels, increasing and decreasing as needed based upon the user’s chosen font scale level in Settings

Dimensions

1. Dimension resources, by default, are held in a dimens.xml file in the res/values/ directory that also holds your strings.
2. To encode a dimension as a resource, add a dimen element to dimens.xml, with a name attribute for your unique name for this resource, and a single child text element representing the value:

```
 <resources>
<dimen name="thin">10dip</dimen>
<dimen name="fat">1in</dimen>
</resources>
```

Weight

1. What happens if we have two or more widgets that should split the available free space?
2. For example, suppose we have two multi-line fields in a column, and we want them to take up the remaining space in the column after all other widgets have been allocated their space.
3. To make this work, in addition to setting android:layout_width (for rows) or android:layout_height (for columns), you must also set android:layout_weight. This property indicates what proportion of the free space should go to that widget.
4. If you set android:layout_weight to be the same non-zero value for a pair of widgets, the free space will be split evenly between them. If you set it to be 1 for one widget and 2 for another widget, the second widget will use up twice the free space that the first widget does. And so on.

Gravity

1. By default, everything in a LinearLayout is left- and top-aligned. So, if you create a row of widgets via a horizontal LinearLayout, the row will start on the left side of the screen.
2. If that is not what you want, you need to specify a gravity. Unlike the physical world, Android has two types of gravity: the gravity of a widget within a LinearLayout, and the gravity of the contents of a widget or container.
3. The android:gravity property of some widgets and containers — which also can be defined via setGravity() in Java — tells Android to slide the contents of the widget or container in a particular direction. For example, android:gravity="right" says to slide the contents of the widget to the right; android:gravity="right|bottom" says to slide the contents of the widget to the right and the bottom.

Sharing information using Intents

The easiest way to do this would be to pass the data or information to the other activity in the `Intent` you're using to start the activity:

```
Intent intent = new Intent(this, ActivityB.class);
intent.putExtra(“YOUR_ID", “value”); startActivity(intent);
```

Access that intent on next activity:

```
String value = getIntent().getStringExtra(“YOUR_ID");
```

Exception Handling

1. **Exception handling** in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.
2. **Exception** is an abnormal event or condition that disrupts the normal flow of the program. Its an object which is thrown at runtime.
3. **Error vs Exception:** Error indicates serious problem that a reasonable application should not try to catch. Exception indicates conditions that a reasonable application might try to catch.
4. **Java exception handling** is managed via five keywords: **try, catch, throw, throws, and finally**. Any **exception** that is thrown out of a method must be specified as such by a throws clause.
5. Any code that absolutely must be executed after a try block completes is put in a catch block.

Exception Handling

```
try{
        Uri uri= Uri.parse("market://details?id="+ getPackageName());
        Intent goToMarket = new Intent(Intent.ACTION_VIEW, uri);
        startActivity(goToMarket);

    }catch(ActivityNotFoundException e){
        Uri uri1= Uri.parse("http://play.google.com/store/apps/details?id="+
getPackageName());
        Intent goToMarket1 = new Intent(Intent.ACTION_VIEW, uri1);
        startActivity(goToMarket1);
    }
```

THANK YOU !