Android Application Development, COMP 10073

Mohawk College, Winter 2021

Toasts

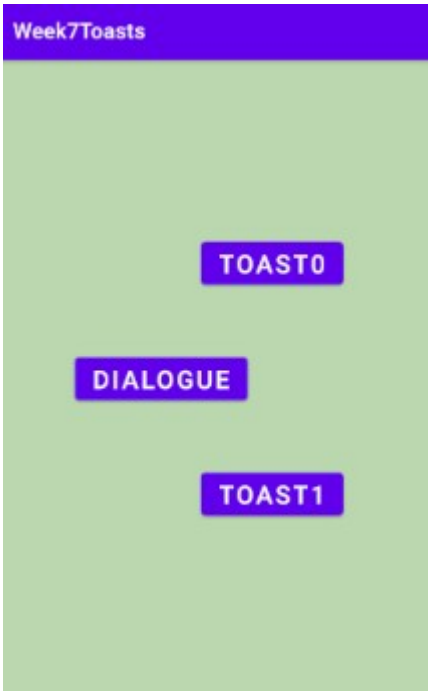
A toast is a popup message that appears briefly on top of an Android Activity. To illustrate start by creating an Android application with several buttons. Label the buttons “toast0”, “toast1” and “dialogue”. The details aren’t very important, we simply need some basic triggers to activate some code. Something like the following will do just fine:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#50238000"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/toast1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/toast1"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/toast0"
        app:layout_constraintTop_toBottomOf="@+id/toast0" />

    <Button
        android:id="@+id/toast0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/toast0"
        android:textSize="24sp"
        app:layout_constraintBottom_toTopOf="@+id/toast1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.7"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/dialogue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:onClick="onClick"
        android:text="dialogue"
        android:textSize="24sp"
        app:layout_constraintBottom_toTopOf="@+id/toast1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.3"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/toast0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



Basic Toast

https://developer.android.com/reference/android/widget/Toast

A toast is a view containing a quick little message for the user. The toast class helps you create and show those.

When the view is shown to the user, appears as a floating view over the application. It will never receive focus. The user will probably be in the middle of typing something else. The idea is to be as unobtrusive as possible, while still showing the user the information you want them to see. Two examples are the volume control, and the brief message saying that your settings have been saved.

The easiest way to use this class is to call one of the static methods that constructs everything you need and returns a new Toast object.

Note that Snackbars are preferred for brief messages while the app is in the foreground.

Add a button handler that responds to the first button, like so:

```
public void onClick(View v) {

    if (v.getId() == R.id.toast0) {
        // chain
        Toast.makeText(this, "Basic Toast", Toast.LENGTH_LONG).show();

    . . .
```

Why not use a string constant here?

Define constants when you are going to reference the same value multiple times. If you are just referencing something as a one off, then constants really don't help.

What is chaining? This is the practice of invoking object methods in a sequence, i.e. using multiple dots. We can write the same code as

```
Toast t2 = Toast.makeText(this, "Unchained =Short=",
        Toast.LENGTH_SHORT);
t2.show();
```

The toast Length argument determins the amount of time the message is visible.

Custom Toasts

Add a drawable border, and a custom layout to the resources. We will use these to make a special custom toast message.

border.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape>
            <corners android:radius="30dp" />
            <stroke android:width="4dp" />
            <solid android:color="#B0FF2090" />
            <padding android:bottom="10dp" android:left="10dp"
                     android:right="10dp" android:top="10dp" />
        </shape>
    </item>
</selector>
```

customlayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/border"
    >
    <AnalogClock
        android:id="@+id/analogClock"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toTopOf="@+id/textView2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="58dp"
        android:text="Current Time"
        android:textSize="30sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/analogClock" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Use the inflator and setView

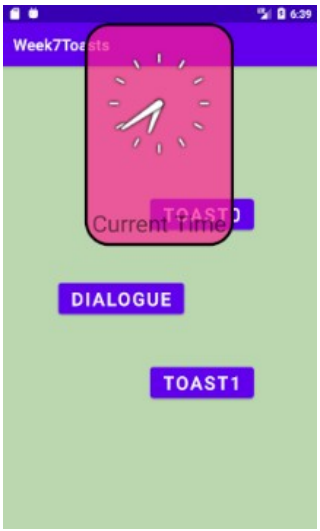
https://developer.android.com/reference/android/view/LayoutInflater

Instantiates a layout XML file into its corresponding View objects. It is never used directly. Instead, use Activity.getLayoutInflater() or Context#getSystemService to retrieve a standard LayoutInflater instance that is already hooked up to the current context and correctly configured for the device you are running on.

For performance reasons, view inflation relies heavily on pre-processing of XML files that is done at build time. Therefore, it is not currently possible to use LayoutInflater with an XmlPullParser over a plain XML file at runtime; it only works with an XmlPullParser returned from a compiled resource (R.something file.)

```
} else if (v.getId() == R.id.toast1) {
    // To show a toast with a custom layout - need to inflate the layout
    LayoutInflater myinflator = getLayoutInflater();
    View customtoast = myinflator.inflate(R.layout.customlayout, null);

    // Now build the Toast - adjust settings
    Toast myToast = new Toast(this);
    // set the view duration
    myToast.setDuration(Toast.LENGTH_LONG);
    // position the toast at the top of the screen
    myToast.setGravity(Gravity.TOP, 0, 0);
    // set connect the custom layout (deprecated)
    myToast.setView( customtoast );
    // And finally show it
    myToast.show();
```



setView is deprecated

https://developer.android.com/reference/android/widget/Toast#setView(android.view.View)

setView

Added in API level 1

Deprecated in API level 30

```
public void setView (View view)
```

This method was deprecated in API level 30.

Custom toast views are deprecated. Apps can create a standard text toast with the `makeText(android.content.Context, java.lang.CharSequence, int)` method, or use a Snackbar when in the foreground. Starting from Android `Build.VERSION_CODES#R`, apps targeting API level `Build.VERSION_CODES#R` or higher that are in the background will not have custom toast views displayed.

Set the view to show.

Snackbars have some properties similar to toasts, we won't cover them this year.

fragment_custom.xml

We can include active widgets in a DialogFragment

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:id="@+id/frameLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/border2"
    tools:context=".CustomDialog">

    <TextView
        android:id="@+id/dialogtext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="27dp"
        android:text="TextView"
        app:layout_constraintBottom_toTopOf="@+id/dialogedit"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_chainStyle="spread_inside" />

    <Button
        android:id="@+id/dialogbutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="9dp"
        android:text="OK"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/dialogedit" />

    <EditText
        android:id="@+id/dialogedit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="27dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/dialogtext" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

border2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape>
            <stroke android:width="10dp" />
            <solid android:color="#A00020F0" />
            <padding android:bottom="50dp" android:left="50dp"
                android:right="50dp" android:top="50dp" />
        </shape>
    </item>
</selector>
```

Sending Data from a DialogFragment

https://developer.android.com/reference/androidx/fragment/app/DialogFragment

You won't find "DialogFragment" in the Android Studio menus. You can simply create the files, or create a Fragment and change the argument.

Use getActivity() to access our main activity. With that reference we can access methods that we define in our main activity.

```
/**
 * For a Dialog we must extend the DialogFragment Class
 * We implement OnClickListener to handle button click events
 */
public class CustomDialog extends DialogFragment
        implements View.OnClickListener {
    public CustomDialog() {
        // Required empty public constructor
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        // We saved the inflated layout in our myview variable
        View myview = inflater.inflate(R.layout.fragment_custom, container, false);

        // Attach the OnClickListener to the button
        Button button = myview.findViewById(R.id.dialogbutton);
        button.setOnClickListener(this);

        // Fill in our title text
        TextView title = myview.findViewById(R.id.dialogtext);
        title.setText("Hello");

        // Don't forget to return the view
        return myview;
    }

    @Override
    public void onClick(View v) {

        // Get the string out of our Edit field
        // getView() will return the view of the current Fragment
        EditText editInput = getView().findViewById(R.id.dialogedit);

        String myInput = editInput.getText().toString();

        // Send the string to the main activity -
        // call a method in the main activity class
        MainActivity main = (MainActivity) getActivity();
        main.sendInput(myInput);

        // Dismiss will close the dialog
        dismiss();
    }
}
```

Receiving the data in an Activity

Methods that we define in the main activity can be accessed from an attached fragment by using getActivity(), and an appropriate cast. Casting is never a great option, but for this example lets try it.

```
public void sendInput(String in) {
    Toast t1 = Toast.makeText(this, in,
            Toast.LENGTH_SHORT);

    t1.show();
}
```

Launching the Custom Dialog

Include the following code in the main activity as part of the button handler, use it to start the custom dialog.

```
// For a dialog, we need to create an instance
CustomDialog myDialog = new CustomDialog();
// And show it.  (Code in the dialog does the rest)
myDialog.show(getSupportFragmentManager(),null);
```