Android: User Interface / Toasts,... and Menus

Ferruccio Damiani

Università di Torino
www.di.unito.it/~damiani

Mobile Device Programming
(Laurea Magistrale in Informatica, a.a. 2018-2019)

Outline

Outline

Toasts [git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/KotlinDialog.git]

- Tiny messages over the Activity
  - Used to highlight minor problems or provide confirmations
  - Usually appear near the bottom of the screen, centered horizontally (gravity)
  - We can control their duration

First, instantiate a Toast object with one of the makeText() methods. This method takes three parameters: the application Context, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with show(), as shown in the following example:

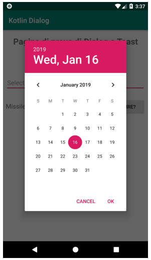
```
    val context = applicationContext
    val text = "Happy Toast!"
    val duration = Toast.LENGTH_SHORT

    Toast.makeText(context, text, duration).show()
```

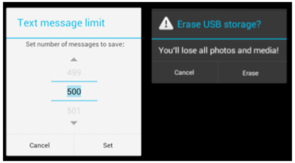
This example demonstrates everything you need for most toast notifications. You should rarely need anything else.[1]

[1]You may, however, want to position the toast differently or even use your own layout instead of a simple text message.

Dialogs

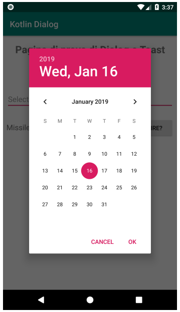
- Used to interact with the user
- Short messages, easy answers
- The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses:
  - AlertDialog: A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
  - ProgressDialog: A dialog showing a progress indicator and an optional text message or view. Only a text message or a view can be used at the same time.
    - ★ ProgressBar would be better
  - DatePickerDialog or TimePickerDialog: A dialog with a pre-defined UI that allows the user to select a date or time.

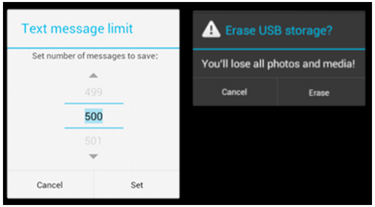


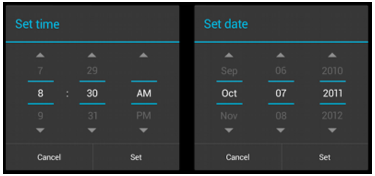

Set number of messages to save:

You'll lose all photos and media!

Examples of dialogs







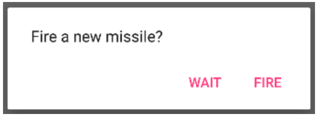
You should use a Dialog (or a DialogFragment) as a container for your dialog.

For example, here's a basic AlertDialog that's managed within a AlertDialog.Builder:

[git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/KotlinDialog.git]

```
    private var alertDialog: Dialog? = null

    fun fireMissilesDialog(view: View) {
        if (alertDialog == null) {
            val builder = AlertDialog.Builder(this).apply {
                setMessage(R.string.missiles_title)
                setPositiveButton(R.string.missiles_ok) { dialog, which ->
                    // FIRE THE MISSILES!
                }
                setNegativeButton(R.string.missiles_no) { _, _ ->
                    // User cancelled the dialog
                }
            }
            alertDialog = builder.create()
        }
        alertDialog!!.show()
    }
```

When you create an instance of this class and call show() on that object, the dialog appears:


Fire a new missile?

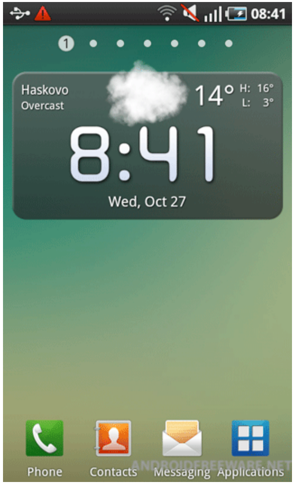
App widgets

Overview

- Widgets are little applications which can be placed on a widget host, typically the home screen or the lock screen
- A widget runs as part of the process of its host
- Android 4.2 introduces the ability for users to add widgets to the lock screen

Limitations

- A widget has the same runtime restrictions as a normal broadcast receiver
  - It only has 5 seconds to finish its processing
- A widget should perform time consuming operations in a service and perform the update of the widget from the service



Animations

- Used to move/shrink/color elements
- Two solutions
  - Subsequent images (frame-by-frame)
  - Initial state, final state, time, transition (tween)
- Android 5.0 (API level 21) includes new APIs to create custom animations in your app
  - Animations are heavy and expensive
  - Too many animations are confusing and may become a problem

Animations: Frame by Frame

- Each Drawable can be a frame of the animation
- Could be defined via XML or in Java
- Based on a particular set of images
  - Same animation on different devices requires different sets
- The .apk becomes bigger and bigger

Animations: Tween

- Defines only the frame of the animation
  - Starts like this, ends like that, and lasts x seconds
- One can define the transparency of objects, their rotation, the dimension of images, and their position on the screen
- The same animation can be applied to multiple objects
  - Animations are independent of the objects they work on

Outline

Menus: three fundamental types[2]

1. Options menu and app bar
   - This is the primary collection of menu items for an activity
   - On Android 3.0 and higher, items from the options menu are presented by the app bar as a combination of on-screen action items and overflow options
2. Context menu and contextual action mode
   - It is a floating menu that appears when the user performs a long-click on an element and provides actions that affect the selected content or context frame
   - On Android 3.0 and higher, you should instead use the contextual action mode to display action items in a bar at the top of the screen and allows the user to select multiple items
3. Popup menu
   - It displays a list of items in a vertical list that is anchored to the view that invoked the menu
     - ★ Good for providing an overflow of actions that relate to specific content

[2]Beginning with Android 3.0 (API level 11), Android powered devices are no longer required to provide a dedicated Menu button. Android apps should provide an app bar to present common user actions.

Menus: how to design and implement them

- Guidelines for properly designing them:
  - If they become too big, they do not fit
  - Should guide the user properly
  - Should exploit the current state/context
- Two ways of implementing them:
  - XML: file in res/menu and inflate it (reuse)
  - Java: directly part of the activity's code

Defining a Menu in XML

For all menu types, Android provides a standard XML format to define menu items.
To define the menu, create an XML file inside your project's res/menu/ directory and build the menu with the following elements:

<menu>	Defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item>and <group> elements.
<item>	Creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu.
<group>	An optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.

- Each menu entry (<item>) is specified by an ID, an icon (optional), and a title. The most important attributes are:

android:id	A resource ID that's unique to the item, which allows the application can recognize the item when the user selects it
android:icon	A reference to a drawable to use as the item's icon.
android:title	A reference to a string to use as the item's title.
android:showAsAction	Specifies when and how this item should appear as an action item in the app bar.

- One can add a submenu to an item in any menu (except a submenu)
- Method onCreateOptionsMenu() inflates the menu
  - Specific actions must then be implemented

```
<menu xmlns:android=...>
       <item android:id="@+id/system"
                 android:icon="@drawable/system"
                 android:title="@string/option1"
                 app:showAsAction="ifRoom"/>
       <item android:id="@+id/about"
                 android:title="@string/about" />
</menu>
```

Creating an Options Menu

[git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/KotlinMenu.git]

Within the activity

- To specify the options menu for an activity, override onCreateOptionsMenu():

```
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        val inflater = menuInflater
        inflater.inflate(R.menu.option_menu, menu)
        return true
    }
```

  You can also add menu items using add() and retrieve items with findItem() to revise their properties with MenuItem APIs.

- To handle click events, override onOptionsItemSelected():

```
    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        return when (item?.itemId) {
            R.id.system -> doSystem()
            R.id.about -> doAbout()
            else -> super.onOptionsItemSelected(item)
        }
    }
```

  When you successfully handle a menu item, return true. If you don't handle the menu item, you should call the superclass implementation of onOptionsItemSelected() (the default implementation returns false).

- To change menu items at runtime, use method onPrepareOptionsMenu()
    - This method passes the Menu object as it currently exists so you can modify it, such as add, remove, or disable items
    - Fragments also provide an onPrepareOptionsMenu()
    - Starting from Android 3.0, when you want to perform a menu update, you must call invalidateOptionsMenu() to request that the system call onPrepareOptionsMenu().[3]

[3] You should never change items in the options menu based on the View currently in focus.

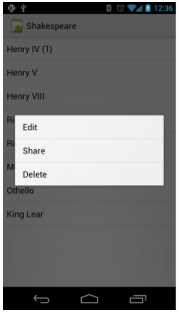
Creating Contextual Menus

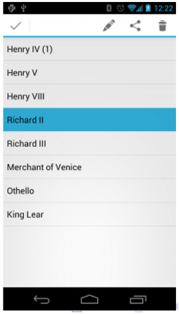
Most often used for items in a ListView, GridView, or other view collections in which the user can perform direct actions on each item. There are two ways to provide contextual actions:

1. In a floating context menu. A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.
2. In the contextual action mode. This mode is a system implementation of ActionMode that displays a contextual action bar at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it).

Screenshots of:

1. a floating context menu (left) and
2. an example of contextual action bar (right).





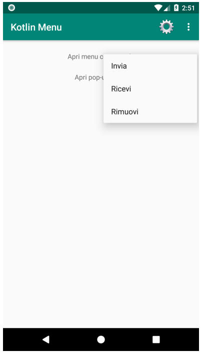
1. Creating a floating context menu [git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/KotlinMenu.git]

- Register the View by calling registerForContextMenu() and pass it the View.
  - If you pass the entire ListView or GridView, you register all of its items
- Implement the onCreateContextMenu() method in your Activity or Fragment

```
    override fun onCreateContextMenu(menu: ContextMenu?, v: View?,
                                       menuInfo: ContextMenu.ContextMenuInfo?) {
        super.onCreateContextMenu(menu, v, menuInfo)
        val inflater = menuInflater
        inflater.inflate(R.menu.context_menu, menu)
    }
```

- Implement onContextItemSelected() to perform the appropriate actions

```
    override fun onContextItemSelected(item: MenuItem?): Boolean {
        // Handle item selection
        return when (item?.itemId) {
            R.id.send -> doSend()
            R.id.recive -> doRecive()
            R.id.remove -> doRemove()
            else -> super.onContextItemSelected(item)
        }
    }
```



2. Using the contextual action mode

- For contextual actions on individual views
  - Implement the ActionMode.Callback interface to specify the actions for the contextual action bar
  - Call startActionMode() when appropriate, such as in response to a long-click on a View
- For batch contextual actions on groups of items in a ListView or GridView
  - The user can select multiple items and perform an action on them all
  - Implement the AbsListView.MultiChoiceModeListener interface and set it for the view group with setMultiChoiceModeListener()
  - Call setChoiceMode() with the CHOICE_MODE_MULTIPLE_MODAL argument

Creating a Popup Menu

A PopupMenu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise. It's useful for:

- Provide an overflow-style menu for actions that relate to specific content
  - This is not the same as a context menu, which is generally for actions that affect selected content
- Provide a second part of a command sentence
  - A button marked "Add" that produces a popup menu with different "Add" options
- Provide a drop-down similar to Spinner that does not retain a persistent selection

If you define your menu in XML, here's how you can show the popup menu:

1. Instantate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.
2. Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu(). On API level 14 and above, you can use PopupMenu.inflate() instead.
3. Call PopupMenu.show().

For example:

[git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/KotlinMenu.git]

- here's a button with the android:onClick attribute that shows a popup menu:

```
<TextView
    android:id="@+id/id_popup"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/popup_menu"
    android:onClick="showPopup" />
```

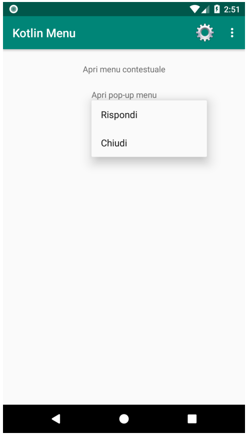
- The activity can then show the popup menu like this:

```
    fun showPopup(view: View) {
        val popup = PopupMenu(this, view)

        popup.inflate(R.menu.popup_menu)
        popup.show()
    }
```

In API level 14 and higher, you can combine the two lines that inflate the menu with PopupMenu.inflate(). The menu is dismissed when the user selects an item or touches outside the menu area. You can listen for the dismiss event using PopupMenu.OnDismissListener.

- To perform an action when the user selects a menu item, you must implement the PopupMenu.OnMenuItemClickListener interface and register it with your PopupMenu by calling setOnMenuItemclickListener(). When the user selects an item, the system calls the onMenuItemClick() callback in your interface.

```
    fun showPopup(view: View) {
        val popup = PopupMenu(this, view)

        // This activity implements OnMenuItemClickListener
        popup.inflate(R.menu.popup_menu)
        popup.setOnMenuItemClickListener(this)
        popup.show()
    }

    override fun onMenuItemClick(item: MenuItem?): Boolean {
        // Handle item selection
        return when (item?.itemId) {
            R.id.popup_start -> doPopupStart()
            R.id.popup_end -> doPopupEnd()
            else -> false
        }
    }
```


2:51
Kotlin Menu
Apri menu contestuale
Apri pop-up menu
Rispondi
Chiudi