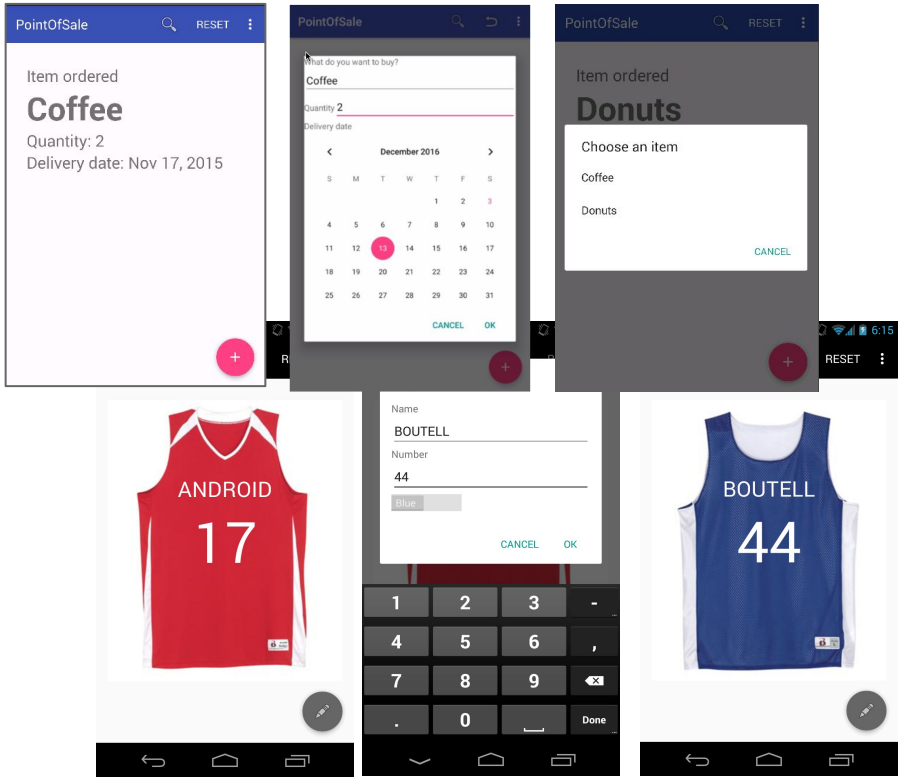
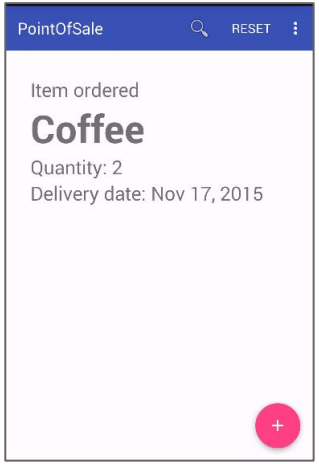
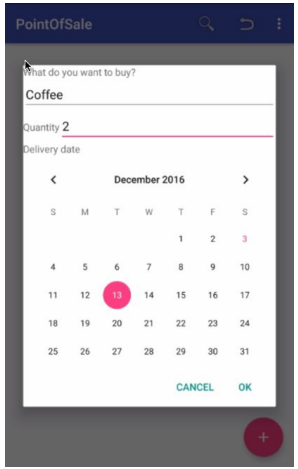
Menus and Dialogs

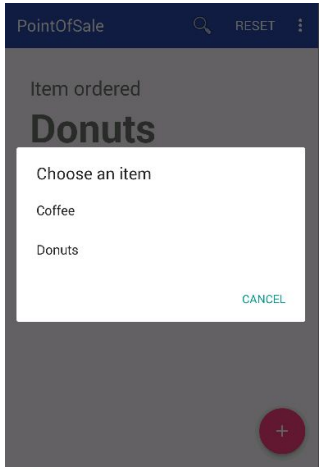


By the end of this unit you should be able to...

- Use the floating action button
- Create toolbar menu items
- Use custom and standard icons
- Create alert dialogs with buttons, lists, and custom views
- Create snackbar messages with actions



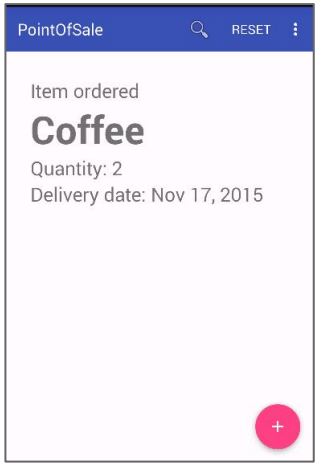




ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Starting code and Floating Action Button

In this lesson you will learn about the floating action button and how to detect its clicks

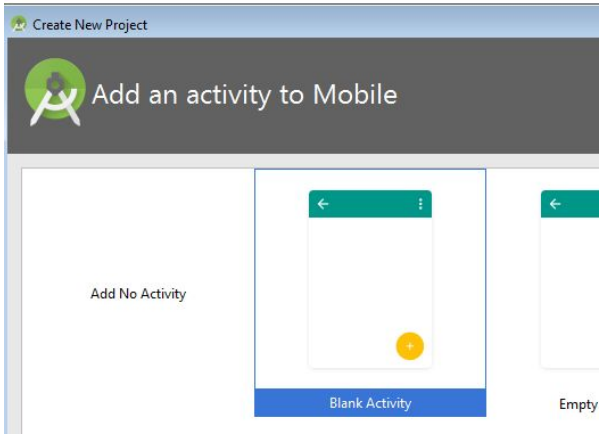


http://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html

Create a new project with a BasicActivity

Name: PointOfSale

Add a **BasicActivity** (not empty activity), so we get a default menu and floating action button.



Check out the starting code

Layout has activity_main has a Coordinator Layout, with toolbar, content_main, and floating action button (FAB).

Add an id to the coordinator layout:

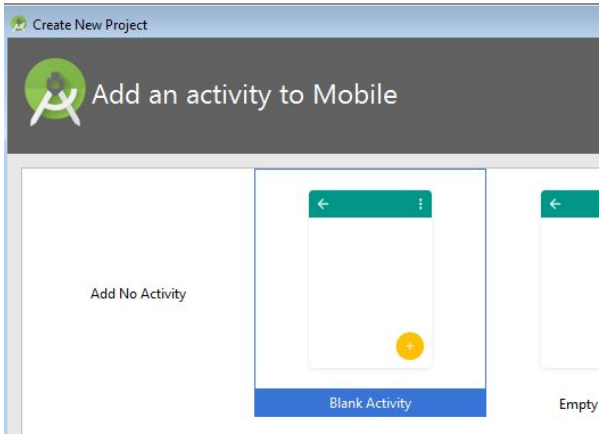
```
android:id="@+id/coordinator_layout"
```

Change the FAB's src icon to

```
android:srcCompat="@drawable/ic_add".
```
(will download)

menu/menu_main.xml has a settings item

MainActivity captures the toolbar & FAB, and has callbacks for the menu.



Download and add these resource files

https://drive.google.com/open?id=1_JeHyWjkhxmLMZ7qbQwzs2CUaC6jfxAY

res/layout/content_main.xml: just 3 TextViews

res/values/strings.xml: headers and format strings

res/drawable/ic_add.png: The + for the FAB

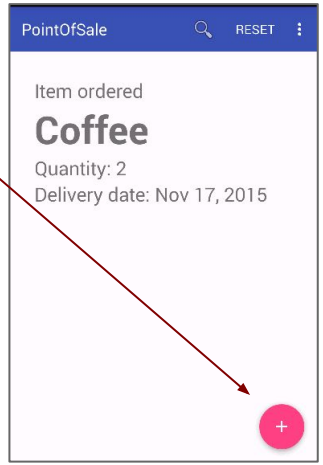
res/dimens.xml: Margins for the layout.

src/Item.java: model file

Move them using your OS to folders in **PointOfSale/app/src/main** if drag and drop doesn't work.

Floating Action Button

"Floating action buttons are used for a special type of promoted action. They are distinguished by a circled icon floating above the UI and have special motion behaviors related to morphing, launching, and the transferring anchor point."



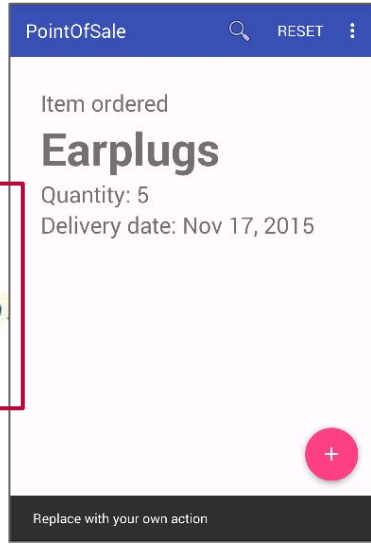
http://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html

Wire up the Floating Action Button to add a default item

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener((view) -> {
            currentItem = Item.getDefaultItem();
            showCurrentItem();
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
    });
```

```java
public void showCurrentItem() {
    mNameText.setText(currentItem.getName());
    mQuantityText.setText(getString(R.string.quantity_format, currentItem.getQuantity()));
    mDateText.setText(getString(R.string.date_format, currentItem.getDeliveryDateString()))
}
```

After click:



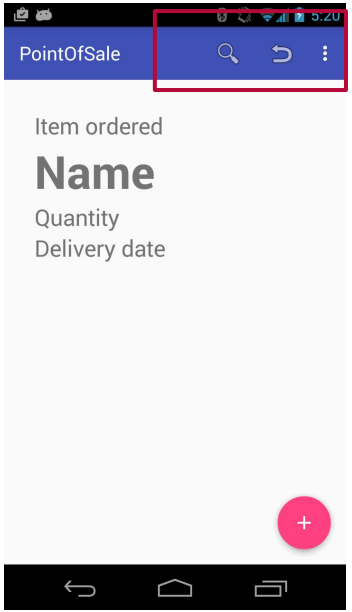
You'll also need to capture the 3 TextViews and store them as fields.

http://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html

Toolbar

In this lesson you will learn how to create and respond to toolbar menu items:

- menu_main.xml
- onCreateOptionsMenu()
- onOptionsItemSelected()

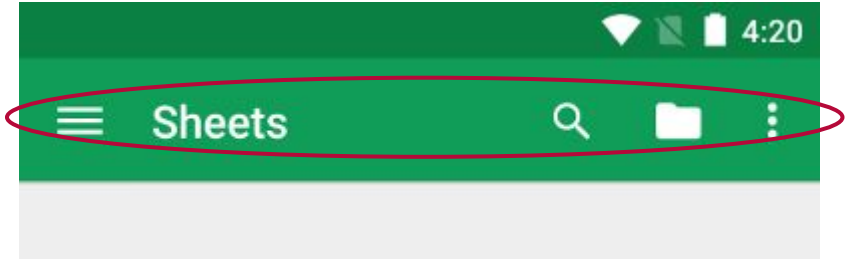


ROSE-HULMAN INSTITUTE OF TECHNOLOGY

The app bar identifies the app and user location, and provides user actions and navigation modes.

Primary goals:

- ❑ Provide a dedicated space for identifying the application brand and user location.
- ❑ Provide consistent navigation and view refinement across different applications.
- ❑ Make key actions for the activity (such as "search", "create", "share", etc.) prominent and accessible to the user in a predictable way.
- ❑ Implemented by a Toolbar (new) or ActionBar (old)



http://developer.android.com/training/appbar/index.html

How can you create toolbar menus? From resources or code.

```
//Resources:
@Override
public boolean onCreateOptionsMenu(Menu menu) {
     super.onCreateOptionsMenu(menu);
     getMenuInflater().inflate(R.menu.main, menu);
     return true;
}
// Programmatically:
@Override
public boolean onCreateOptionsMenu(Menu menu) {
     super.onCreateOptionsMenu(menu);
     MenuItem incMenuItem = menu.add(0, ITEM_ID_INCREMENT, Menu.NONE,
     R.string.increment);
     MenuItem decMenuItem = menu.add(0, ITEM_ID_DECREMENT, Menu.NONE,
     R.string.decrement);
     incMenuItem.setIcon(R.drawable.add);
     decMenuItem.setIcon(R.drawable.remove);
     return true;
}
```

Using a resource is better MVC practice

Google's example: res/menu/options_menu.xml

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:title="@string/item1"
          android:icon="@drawable/group_item1_icon"
          android:showAsAction="ifRoom|withText"/>
    <group android:id="@+id/group">
        <item android:id="@+id/group_item1"
              android:onClick="onGroupItemClick"
              android:title="@string/group_item1"
              android:icon="@drawable/group_item1_icon" />
        <item android:id="@+id/group_item2"
              android:onClick="onGroupItemClick"
              android:title="@string/group_item2"
              android:icon="@drawable/group_item2_icon" />
    </group>
    <item android:id="@+id/submenu"
          android:title="@string/submenu_title"
          android:showAsAction="ifRoom|withText" >
        <menu>
            <item android:id="@+id/submenu_item1"
                  android:title="@string/submenu_item1" />
        </menu>
    </item>
</menu>
```

Note: items and groups, even submenus (nested menus)

http://developer.android.com/guide/topics/resources/menu-resource.html

Make a menu item to reset the current Item

Note each menu item has id, icon, title, and when to show it. Add one to reset the current item

```
<menu 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"
    tools:context=".MainActivity">

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />

    <item
        android:id="@+id/action_reset"
        android:icon="@android:drawable/ic_menu_revert"
        android:orderInCategory="100"
        android:title="@string/reset"
        app:showAsAction="ifRoom" />
```

Three menu callbacks in code

```
public boolean onCreateOptionsMenu(Menu menu)
```

When the menu button is first pressed, this function is called to create the items in the initial menu.

```
public boolean onOptionsItemSelected(MenuItem item)
```

When an item in the menu is selected, this function is the click listener.

```
public boolean onPrepareOptionsMenu(Menu menu)
```

Every time the menu button is pressed, this function is called just before the menu is displayed to make changes to account for the current activity state

Note the given callbacks

```
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
```

Return true: we handled it. Don't pass it to other handlers.

Note the given callbacks, and extend them to listen for our new menu item

I use switch statements if I'm going to have several menu items.

Respond to the reset item.

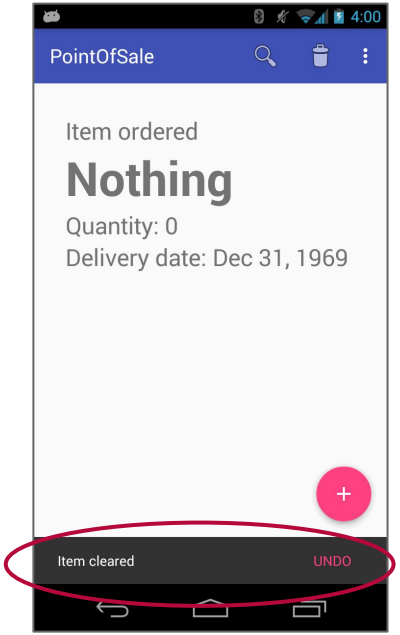
Also launch the system settings if chosen.

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    switch (item.getItemId()) {
        case R.id.action_reset:
            mCurrentItem = new Item();
            showCurrentItem();
            return true;

        case R.id.action_settings:
            startActivity(new Intent(Settings.ACTION_SETTINGS));
            return true;
    }

    return super.onOptionsItemSelected(item);
}
```

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Snackbar

In this lesson you will learn how to work with Android's Snackbar

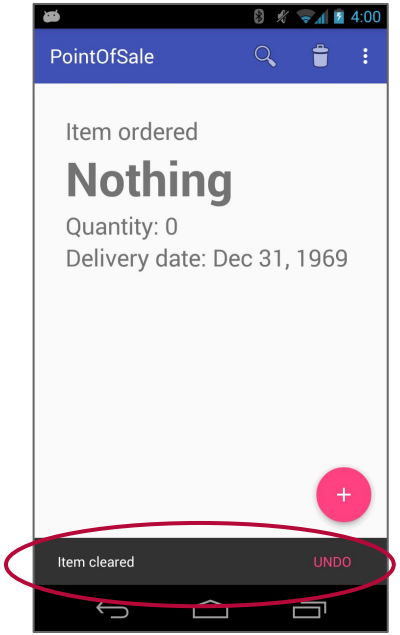


ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Snackbars are for feedback

Snackbars provide lightweight feedback about an operation by showing a brief message at the bottom of the screen. Snackbars can contain an action.

They are like a Toast, but can be swiped away and can contain actions.

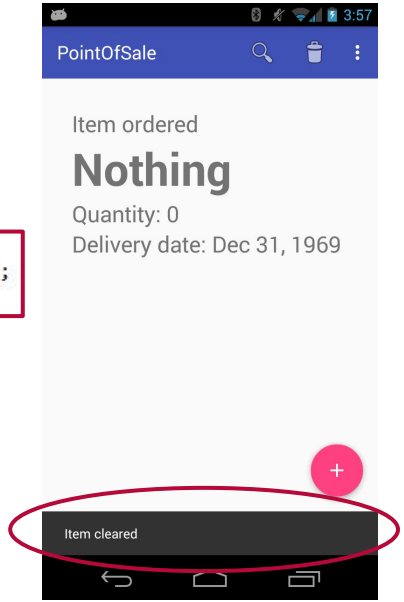


https://www.google.com/design/spec/components/snackbars-toasts.html#

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Add a Snackbar to show when an item has been reset

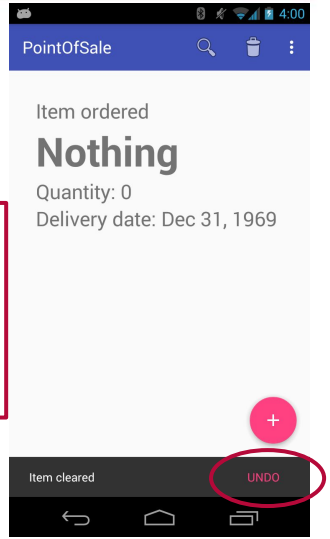
```java
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    switch (item.getItemId()) {
        case R.id.action_reset:
            mClearedItem = mCurrentItem;
            mCurrentItem = new Item();
            showCurrentItem();
            Snackbar snackbar = Snackbar
                    .make(findViewById(R.id.coordinator_layout), "Item cleared", Snackbar.LENGTH_LONG);
            snackbar.show();
            return true;

        case R.id.action_settings:
            startActivity(new Intent(Settings.ACTION_SETTINGS));
            return true;
    }

    return super.onOptionsItemSelected(item);
}
```



Note that the first parameter is the root view. We added its id earlier in this unit.

Even cooler: provide undo functionality via an action!

```
switch (item.getItemId()) {
    case R.id.action_reset:
        mClearedItem = mCurrentItem;
        mCurrentItem = new Item();
        showCurrentItem();
        Snackbar snackbar = Snackbar
                .make(findViewById(R.id.coordinator_layout), "Item cleared", Snackbar.LENGTH_LONG)
                .setAction("UNDO", new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        mCurrentItem = mClearedItem;
                        mClearedItem = null;
                        showCurrentItem();
                        Snackbar.make(findViewById(R.id.coordinator_layout), "Item is restored", Snackbar.LENGTH_LONG).show();
                    }
                });
        snackbar.show();
        return true;
```

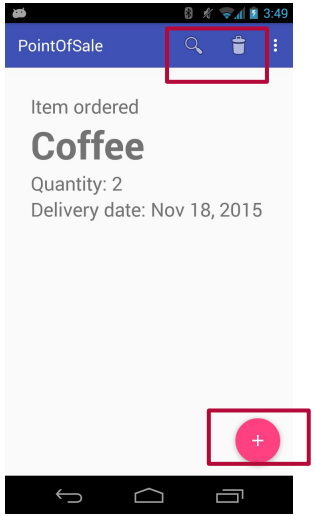


Save off the current item so we can restore it.

http://www.androidhive.info/2015/09/android-material-design-snackbar-example/

Iconography

In this lesson you will learn how to create custom icons and where to find Android's standard ones



There are multiple sources of Android icons

Built-in

Just type `@android:drawable/` where needed and browse

Guidelines

https://www.google.com/design/spec/style/icons.html#icons-system-icons

Icon downloads

https://www.google.com/design/icons/index.html

Android Asset Studio to make custom icons from images/text/clipart

https://romannurik.github.io/AndroidAssetStudio/

Android Asset Studio to the rescue!

Handy tool for creating icons of all different sizes.

Input: **transparent image**
Output: **4 drawable folders**, one for each resolution, standard naming

May be helpful when working on your screen layouts for your next project deliverable?

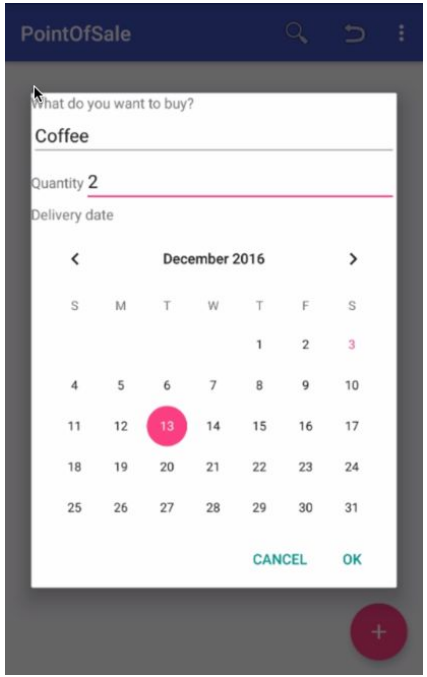
There are multiple sources of Android icons

This is a sampling of the old icons

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Dialogs

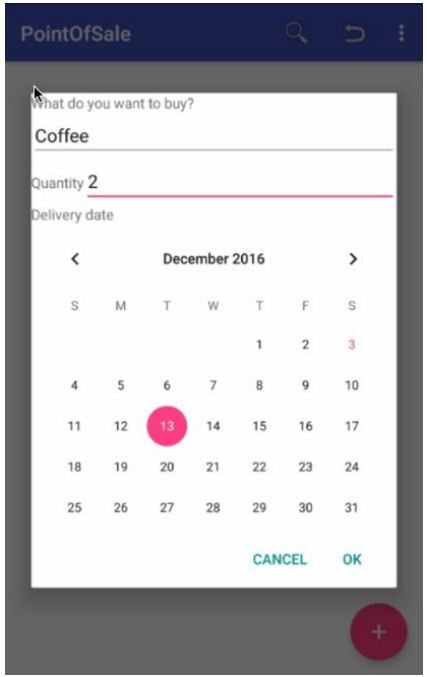
In this lesson you will learn about AlertDialogs



What is a dialog in Android?

A dialog is usually a small window that appears in front of the current Activity. The underlying Activity loses focus and the dialog accepts all user interaction. Dialogs are normally used for notifications and short activities that directly relate to the application in progress.

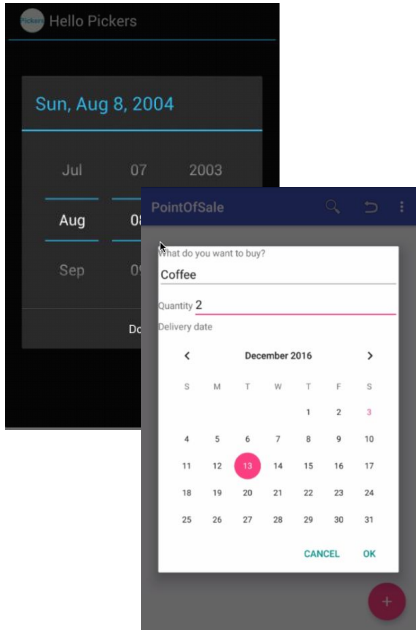
Note: while Activities can be given a dialog theme, actual dialogs are much lighter weight



Dialog classes

AlertDialog

A dialog that can manage zero, one, two, or three buttons, and/or a list of selectable items that can include checkboxes or radio buttons. **The AlertDialog is capable of constructing most dialog user interfaces and is the suggested dialog type.**



Native Dialogs for specific tasks (learn on your own if needed)

- DatePickerDialog
  A dialog to select a date. See the Hello DatePicker tutorial.
- TimePickerDialog
  A dialog to select a time. See the Hello TimePicker tutorial.
- ProgressDialog
  A dialog that displays a progress wheel or progress bar.

Customize your own dialog

Extend the base Dialog object and define a new custom layout.

http://developer.android.com/reference/android/app/AlertDialog.html, pre-2014

Let’s call and write an addItem() method to show a dialog to get info for a new Item

```
        FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                addItem(); // arbitrary name; we’ll write it
//                mCurrentItem = Item.getDefaultItem();
//                mItems.add(mCurrentItem);
//                showCurrentItem();
            }
        });
```

Create a DialogFragment that overrides onCreateDialog()

```
protected void addItem() {
    DialogFragment df = new DialogFragment() {
        @Override
        public Dialog onCreateDialog(Bundle b)

            // TODO
        }

    .show(getFragmentManager(), "");
}
```

UPDATE: Don't use DialogFragments for simple apps. We'll use the guts of onCreateDialog, though. See next slides.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

AlertDialog.Builder

AlertDialog.Builder(Context context):
Constructor using a context for this builder and the AlertDialog it creates.

setIcon(int iconId)

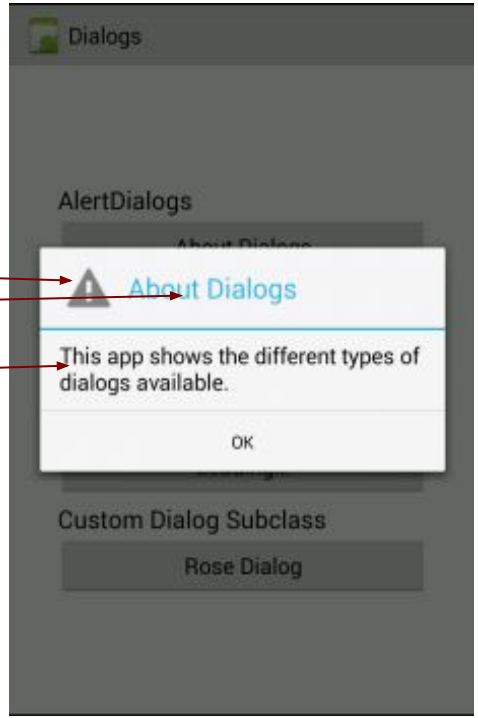
setTitle(CharSequence title)

setMessage(CharSequence message)

setView(View): for custom views!

... (many more **set** methods) ...

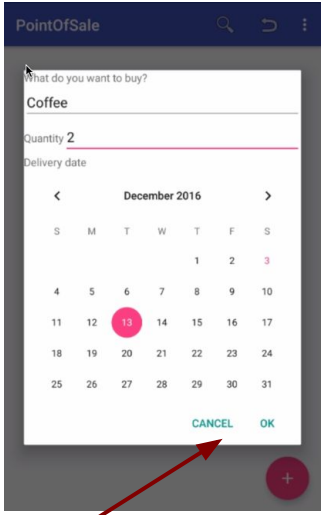
create() creates an AlertDialog with the arguments supplied to this builder.
show() shows the Dialog.



Structure of simple addEdit() that uses no DialogFragment

```
private void addItem() {
   AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
   // Set builder options.
   ...
   builder.create().show();
}
```

AlertDialog.Builder with a custom view



setView(View): for custom views!

We'll need to make this layout resource.

Plan it now.

What widgets does it seem to use?

```
import android.support.v7.app.AlertDialog;
```

to get the nice Material Design borderless button style

dialog_add.xml for reference or copy-paste

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <TextView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="What do you want to buy?" />
   <EditText
       android:id="@+id/edit_name"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:hint="Item name"
       android:inputType="textCapSentences" />
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="horizontal">
       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="@string/quantity_start" />
```

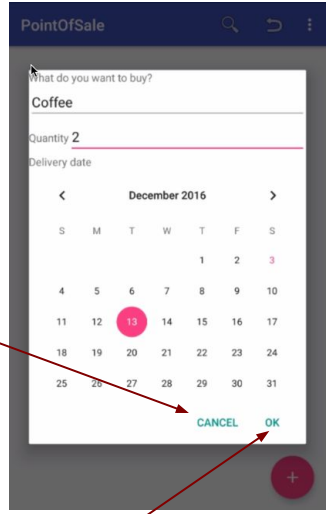
```
(continued from left)
<EditText
    android:id="@+id/edit_quantity"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="1"
    android:inputType="number" />
</LinearLayout>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Delivery date" />
<CalendarView
    android:id="@+id/calendar_view"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:showWeekNumber="false"
    />
</LinearLayout>
```

http://developer.android.com/reference/android/widget/CalendarView.html

AllowDialogs allow options for three buttons

```
setNegativeButton(CharSequence text,
DialogInterface.OnClickListener listener)
```

Set a listener to be invoked when the negative button (**cancel**) of the dialog is pressed.




```
setNeutralButton(CharSequence text, DialogInterface.OnClickListener
listener)
```

...when the neutral button of the dialog is pressed (optional).

```
setPositiveButton(CharSequence text,
DialogInterface.OnClickListener listener)
```

...when the positive button (**ok**) of the dialog is pressed.

Use AlertDialog.Builder to specify the dialog properties, then create the Dialog from it.

Type/copy this in:

1. Make a builder
2. Inflate and set view.
3. Capture parts of view
4. Set the OK button to grab the selections
5. Create a dialog

```
private void addItem() {
    AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
    View view = getLayoutInflater().inflate(R.layout.dialog_add, null, false);
    builder.setView(view);
    final EditText nameEditText = (EditText) view.findViewById(R.id.edit_name);
    final EditText quantityEditText = (EditText) view.findViewById(R.id.edit_quantity);
    final CalendarView deliveryDateView = (CalendarView) view.findViewById(R.id.calendar_view);
    final GregorianCalendar calendar = new GregorianCalendar();
    deliveryDateView.setOnDateChangeListener(new CalendarView.OnDateChangeListener() {
      @Override
      public void onSelectedDayChange(CalendarView view, int year, int month, int dayOfMonth) {
          calendar.set(year, month, dayOfMonth);
      }
    });
    builder.setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
          String name = nameEditText.getText().toString();
          int quantity = Integer.parseInt(quantityEditText.getText().toString());
          mCurrentItem = new Item(name, quantity, calendar);
          mItems.add(mCurrentItem);
          showCurrentItem();
      }
    });
    builder.setNegativeButton(android.R.string.cancel, null);
    builder.create().show();
```

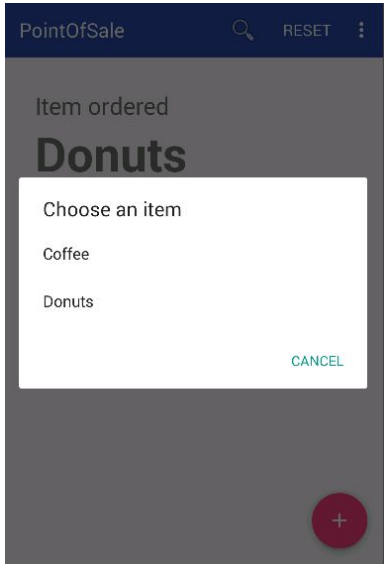
This Calendar code is updated from video's

You don't find these in the Activity, but in this dialog. activity.findViewById() won't find it, the button will be null and a null pointer exception on the next line.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

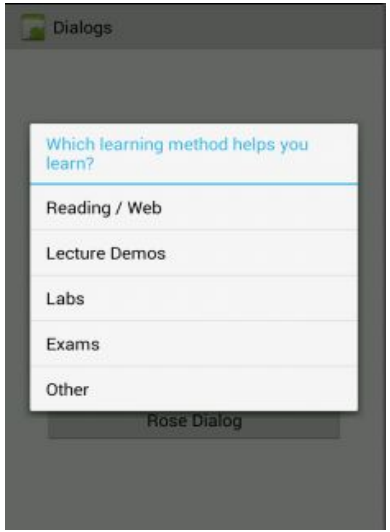
Alert Dialogs with multiple choices

In this lesson you will learn how to provide and detect multiple choices in your dialogs

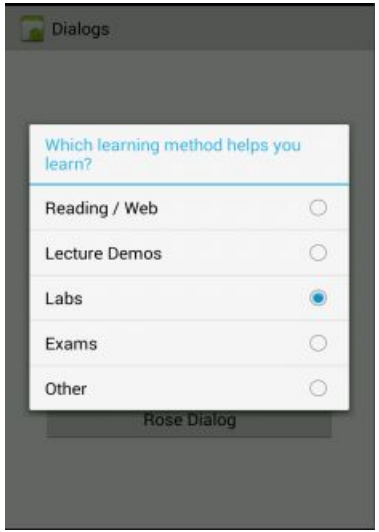


Dialogs allow more than buttons!

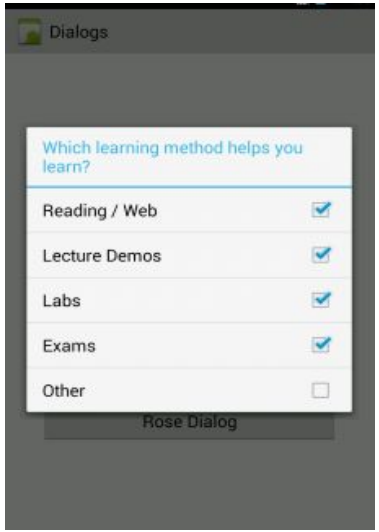
Three types of lists



List



Radio Buttons



Check boxes

Methods for List / Radio / Checkbox buttons

To show a list:

```
setItems(CharSequence[] items, DialogInterface.OnClickListener listener)
```

To show RadioButtons:

```
setSingleChoiceItems(CharSequence[] items, int checkedItem,
DialogInterface.OnClickListener listener)
```

To show checkboxes:

```
setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems,
DialogInterface.OnMultiChoiceClickListener listener)
```

For all, you will be notified of the selected item via the supplied listener.

(all can also accept an array resource instead of a CharSequence[])

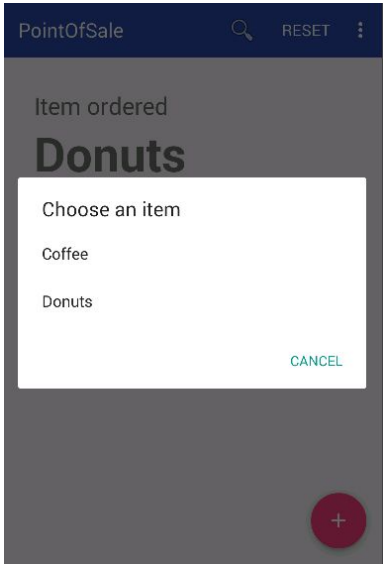
Our goal is to display any added item that is selected by the user

Create the search menu item:

1. menu.xml: Add a new item. Use an appropriately-named @android:drawable/ic_menu_??? icon.

2. .java: When selected, have it call a yet-to-be-written showSearchDialog() method

Do it now.



Create a dialog to show a custom list containing the names of each item. It requires a String[].

Don't also setMessage(). It won't show the list of items!

```
private void showSearchDialog() {
        AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setTitle("Choose an item");
        builder.setItems(getNames(), new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                mCurrentItem = mItems.get(which);
                showCurrentItem();
            }
        });
        builder.setNegativeButton(android.R.string.cancel, null);
        builder.create().show();
```

```
private String[] getNames() {
    String[] names = new String[mItems.size()];
    for (int i = 0; i < mItems.size(); i++) {
        names[i] = mItems.get(i).getName();
    }
    return names;
}
```

```
}
```

This simpler version skips the DialogFragment so doesn't crash

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

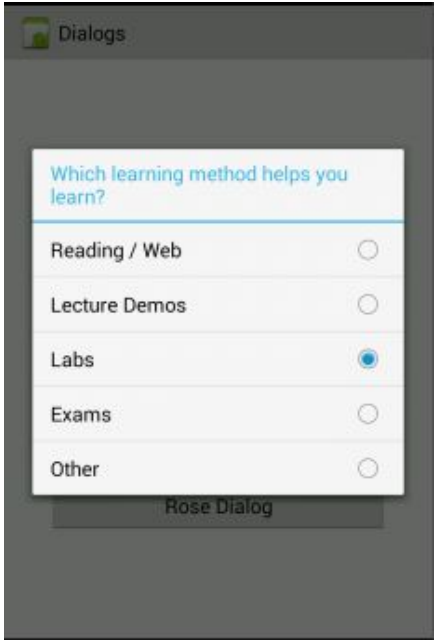
For radio buttons, setSingleChoiceItems()

Instead of setItems()

You may experiment with the radio button version if you like.

Need to set which one is initially selected

Use the same Listener

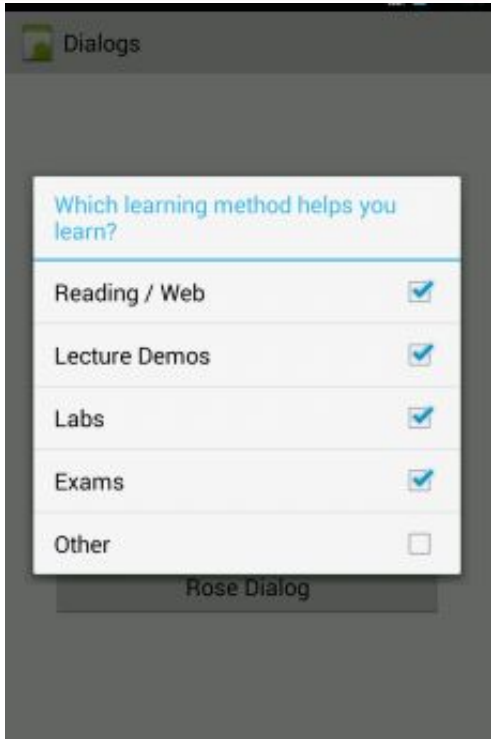


For checkboxes, use setMultichoiceItems()

An array of booleans shows which are initially checked.

Uses a different listener

on**MultiChoice**ClickListener



Checkboxes

```
surveyBuilder.setMultiChoiceItems(
        R.array.learning_method_survey, new boolean[] { true,
                true, true, true, false },
        new DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog,
                    int which, boolean isChecked) {
                String items[] = getActivity().getResources()
                        .getStringArray(
                                R.array.learning_method_survey);
                Toast.makeText(getActivity(),
                        "Yeah for " + items[which] + "!",
                        Toast.LENGTH_SHORT).show();
                dialog.dismiss();
            }
        });
```

Use the isChecked parameter to tell whether the item was selected or unselected.

```
final String[] items = getResources().getStringArray(R.array.learning_methods_array);
builder.setMultiChoiceItems(items,
        new boolean[] {true, false, false, true, false},
        new DialogInterface.OnMultiChoiceClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which, boolean isChecked) {
        Toast.makeText(getActivity(),
                (isChecked ? "Yeah for " : "Boo for ") + items[which],
                Toast.LENGTH_SHORT).show();
    }
});
```

Note: We could add a 'Done' button instead of dismissing each time.

Confirmation dialogs

In this lesson you will practice creating confirmation dialogs

Goal: Add a menu item to clear *all* items entered

This is a big deal, so we want to make sure the user is sure.

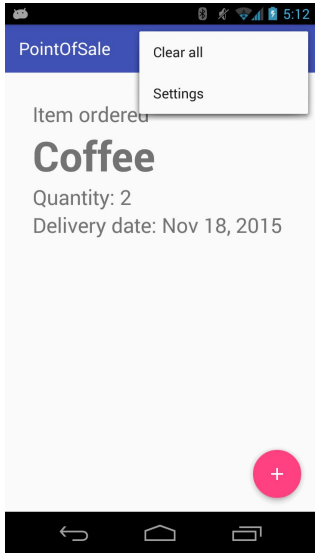
Create a menu item. I did `app:showAsAction="never"` so it always appears in the overflow menu:

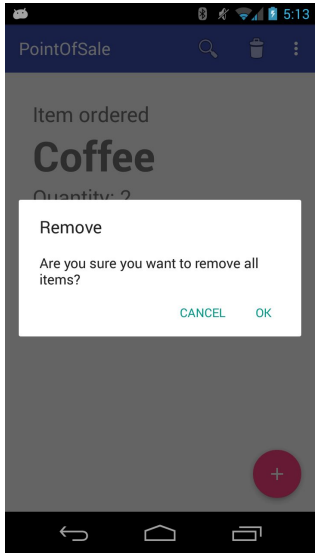
Have it launch a showConfirmationDialog like this:

This doesn't have a custom view, just a title, a message, and positive and negative buttons.

If they choose "OK", it will clear the ArrayList.

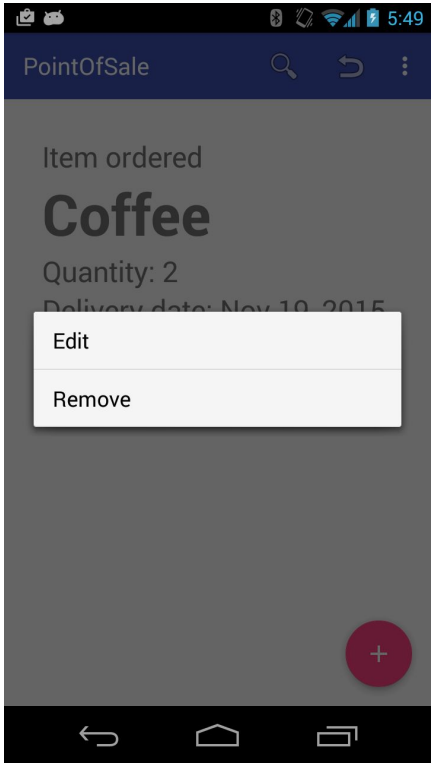
Review the lessons earlier in this unit as needed to do this.





Floating context menus

In this lesson you will learn how to create floating context menus

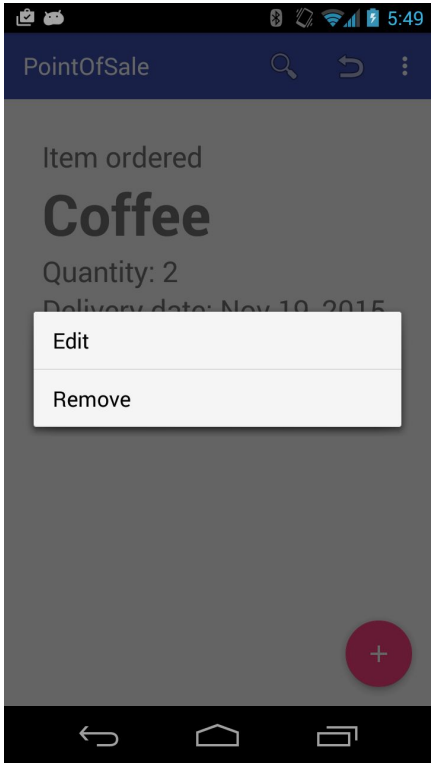


http://developer.android.com/guide/topics/ui/menus.html#FloatingContextMenu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Our goal: to provide multiple actions based on a selection

Click on the name to have the option to edit the item or delete it from the history.



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Step 1: register the context menu

In MainActivity's onCreate():

```
registerForContextMenu(mNameText);
```

I chose to listen for presses on the **name text** since it was the largest font.

Step 2: Just like option menus: create the context menu in xml and inflate it.

```java
// Very similar to options menus!
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   getMenuInflater().inflate(R.menu.menu_context, menu);
}
```

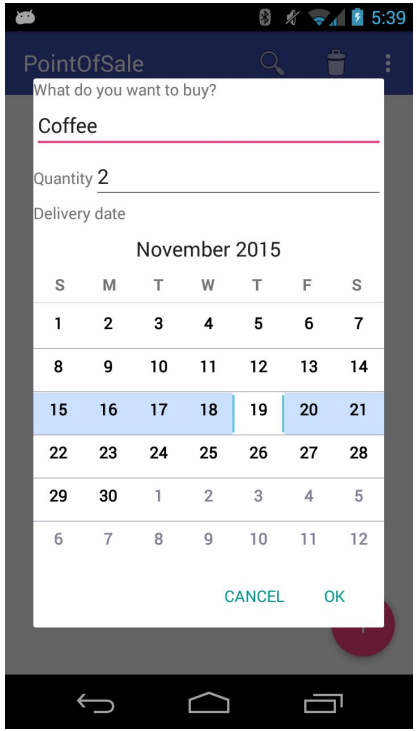
```xml
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/androi
d"
   xmlns:tools="http://schemas.android.com/tools"
   tools:context=".MainActivity">

   <item
       android:id="@+id/menu_context_edit"
       android:icon="@android:drawable/ic_menu_edit"
       android:showAsAction="ifRoom"
       android:title="@string/edit"
       tools:ignore="AppCompatResource" />
   <item
       android:id="@+id/menu_context_remove"
       android:icon="@android:drawable/ic_menu_delete"
       android:showAsAction="ifRoom"
       android:title="@string/remove"
       tools:ignore="AppCompatResource" />
</menu>
```

Step 3: Listen for clicks and respond

When it is clicked, we want to launch a dialog to edit the current item.

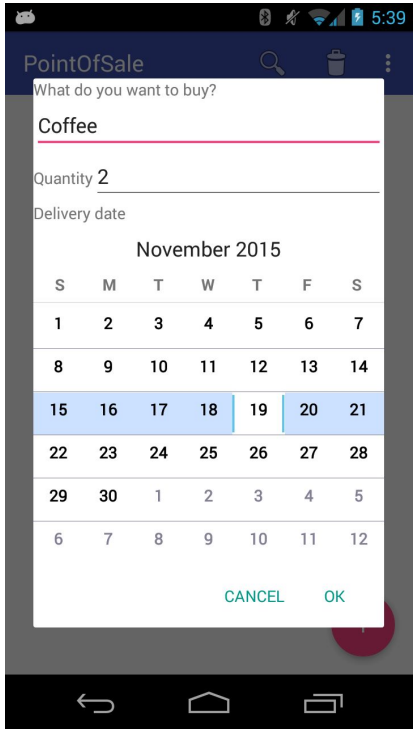
Wait, this is just like the add dialog, except that we are prepopulating the widgets with the current item's data!

Idea: refactor the addItem() method to make it addEditItem(boolean isEditing)



Here is how we'll respond to edit and remove

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_context_edit:
            addEditItem(true);
            return true;
        case R.id.menu_context_remove:
            mItems.remove(mCurrentItem);
            mCurrentItem = new Item();
            showCurrentItem();
            return true;
    }
    return super.onContextItemSelected(item);
}
```



Refactored addEditItem(), pre-populating

```
private void addEditItem(final boolean isEditing) {
    DialogFragment df = new DialogFragment() {
        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
            // Inside onCreateDialog

            AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
            View view = getActivity().getLayoutInflater().inflate(R.layout.dialog_add, null, false);
            builder.setView(view);
            final EditText nameEditText = (EditText) view.findViewById(R.id.edit_name);
            final EditText quantityEditText = (EditText) view.findViewById(R.id.edit_quantity);
            final CalendarView deliveryDateView = (CalendarView) view.findViewById(R.id.calendar_view);
            if (isEditing) {
                nameEditText.setText(mCurrentItem.getName());
                quantityEditText.setText(mCurrentItem.getQuantity() + "");
                deliveryDateView.setDate(mCurrentItem.getDeliveryDateTime());
            }
            builder.setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
```

You'll need to pass false when adding an item

Refactored addEditItem(), mutation

```
builder.setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        // Todo
        String name = nameEditText.getText().toString();
        int quantity = Integer.parseInt(quantityEditText.getText().toString());
        long deliveryDate = deliveryDateView.getDate();
        if (isEditing) {
            mCurrentItem.setName(name);
            mCurrentItem.setQuantity(quantity + “”);
            mCurrentItem.setDeliveryDate(new Date(deliveryDate));
        } else {
            mCurrentItem = new Item(name, quantity, new Date(deliveryDate));
            mItems.add(mCurrentItem);
        }

        showCurrentItem();
    }
});
```

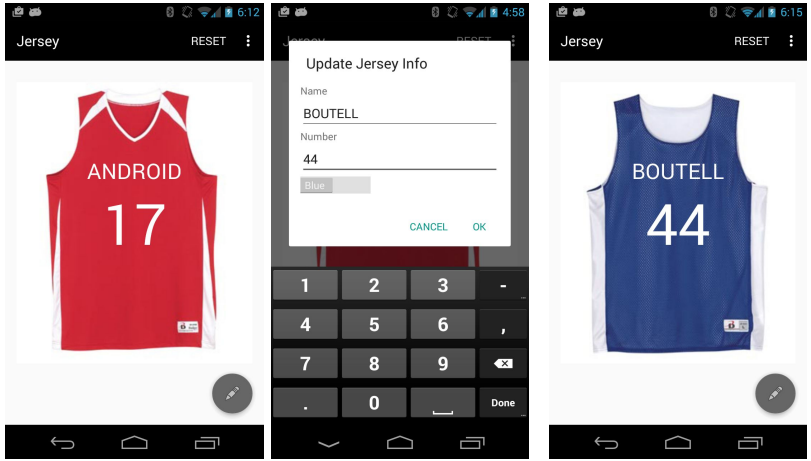
Mutate the current item, don’t add it.

Lab: Jersey

Your turn!

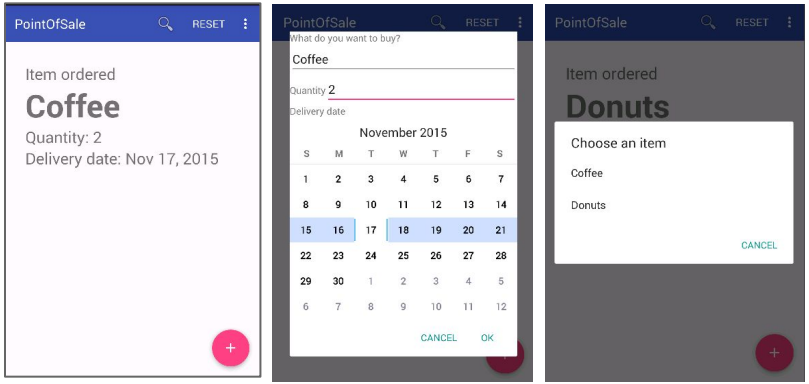
Many menus and dialogs.

Plus Portuguese language support and SharedPreferences to persist simple data.



Summary: menus and dialogs provide for interaction within an activity

You learned about the FAB, the toolbar with menu items, context menus, the snackbar, and AlertDialogs.



For more, especially instructions on how to treat dialogs as activities and vice- versa depending on screen size, see http://developer.android.com/guide/topics/ui/dialogs.html