

Lesson 4.2: Input controls

Introduction

To enable the user to enter text or numbers, you use an [EditText](#) element. Some input controls are EditText attributes that define the type of keyboard that appears, to make entering data easier for users. For example, you might choose phone for the [android:inputType](#) attribute to show a numeric keypad instead of an alphanumeric keyboard.

Other input controls make it easy for users to make choices. For example, [RadioButton](#) elements enable a user to select one (and only one) item from a set of items.

In this practical, you use attributes to control the on-screen keyboard appearance, and to set the type of data entry for an EditText. You also add radio buttons to the DroidCafe app so the user can select one item from a set of items.

What you should already know

You should be able to:

- Create an Android Studio project from a template and generate the main layout.
- Run apps on the emulator or a connected device.
- Create and edit UI elements using the layout editor and XML code.
- Access UI elements from your code using [findViewById\(\)](#).
- Convert the text in a View to a string using [getText\(\)](#).
- Create a click handler for a Button click.
- Display a Toast message.

What you'll learn

- How to change the input methods to enable suggestions, auto-capitalization, and password obfuscation.
- How to change the generic on-screen keyboard to a phone keypad or other specialized keyboards.
- How to add radio buttons for the user to select one item from a set of items.
- How to add a spinner to show a drop-down menu with values, from which the user can select one.

What you'll do

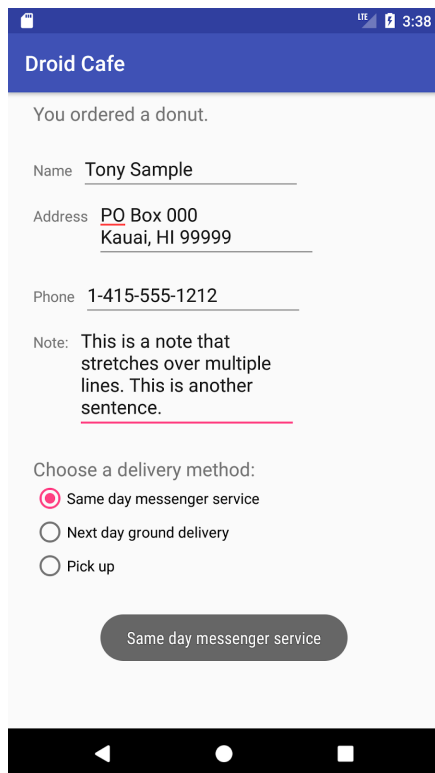
- Show a keyboard for entering an email address.
- Show a numeric keypad for entering phone numbers.
- Allow multiple-line text entry with automatic sentence capitalization.
- Add radio buttons for selecting an option.
- Set an `onClick` handler for the radio buttons.
- Add a spinner for the phone number field for selecting one value from a set of values.

App overview

In this practical, you add more features to the DroidCafe app from the lesson on using clickable images.

In the app's OrderActivity you experiment with the `android:inputType` attribute for `EditText` elements. You add `EditText` elements for a person's name and address, and use attributes to define single-line and multiple-line elements that make suggestions as you enter text. You also add an `EditText` that shows a numeric keypad for entering a phone number.

Other types of input controls include interactive elements that provide user choices. You add radio buttons to DroidCafe for choosing only one delivery option from several options. You also offer a spinner input control for selecting the label (**Home**, **Work**, **Other**, **Custom**) for the phone number.



Task 1: Experiment with text entry attributes

Touching an [EditText](#) editable text field places the cursor in the text field and automatically displays the on-screen keyboard so that the user can enter text.

An editable text field expects a certain type of text input, such as plain text, an email address, a phone number, or a password. It's important to specify the input type for each text field in your app so that the system displays the appropriate soft input method, such as an on-screen keyboard for plain text, or a numeric keypad for entering a phone number.

1.1 Add an EditText for entering a name

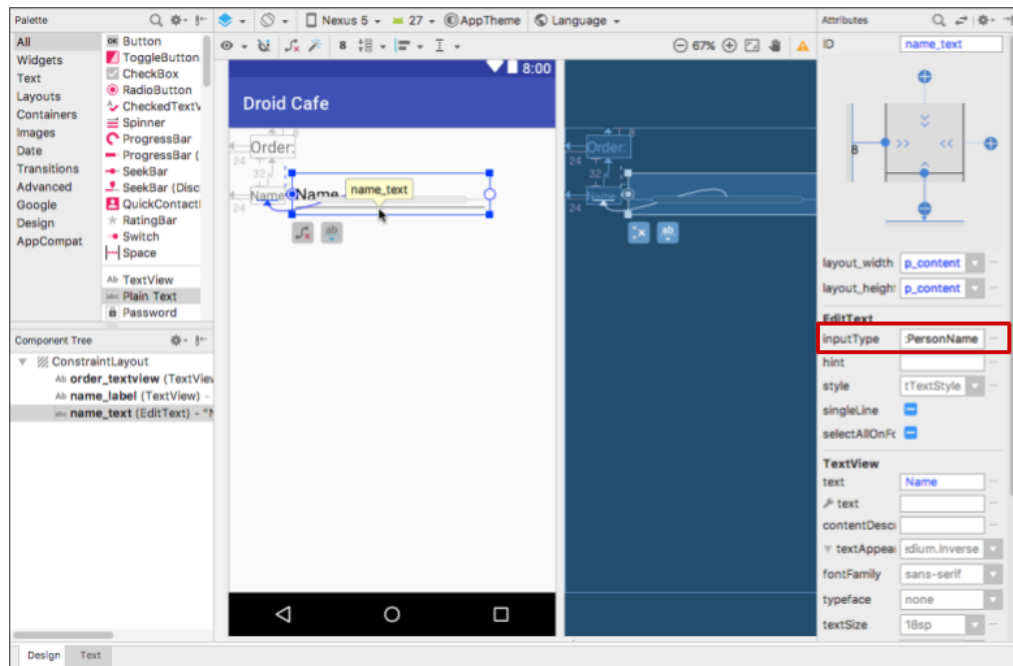
In this step you add a `TextView` and an [EditText](#) to the `OrderActivity` layout in the `DroidCafe` app so that the user can enter a person's name.

1. Make a copy of the `DroidCafe` app from the lesson on using clickable images, and rename the copy to **DroidCafeInput**. If you didn't complete the coding challenge in that lesson, download the [DroidCafeChallenge](#) project and rename it to **DroidCafeInput**.
2. Open the **activity_order.xml** layout file, which uses a `ConstraintLayout`.
3. Add a `TextView` to the `ConstraintLayout` in `activity_order.xml` under the `order_textview` element already in the layout. Use the following attributes for the new `TextView`:

TextView attribute	Value
<code>android:id</code>	<code>"@+id/name_label"</code>
<code>android:layout_width</code>	<code>"wrap_content"</code>
<code>android:layout_height</code>	<code>"wrap_content"</code>
<code>android:layout_marginStart</code>	<code>"24dp"</code>
<code>android:layout_marginLeft</code>	<code>"24dp"</code>

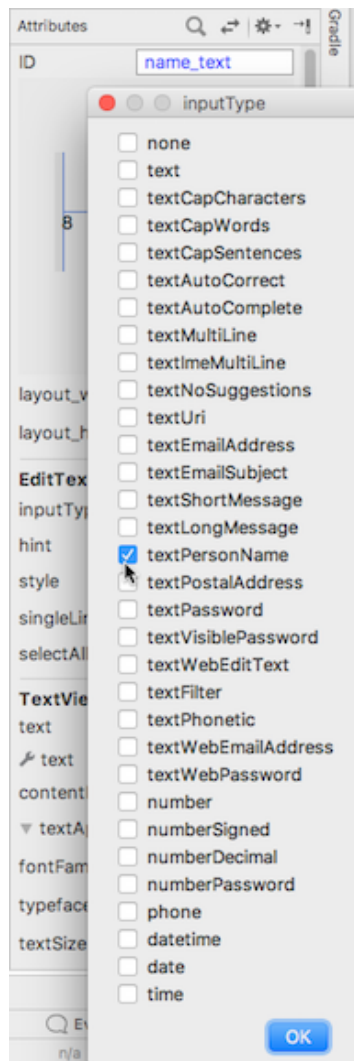
android:layout_marginTop	"32dp"
android:text	"Name"
app:layout_constraintStart_toStartOf	"parent"
app:layout_constraintTop_toBottomOf	"@+id/order_textview"

4. Extract the string resource for the android:text attribute value to create an entry for it called name_label_text in strings.xml.
5. Add an EditText element. To use the visual layout editor, drag a **Plain Text** element from the **Palette** pane to a position next to the name_label TextView. Then enter **name_text** for the **ID** field, and constrain the left side and baseline of the element to the name_label element right side and baseline as shown in the figure below:



6. The figure above highlights the **inputType** field in the **Attributes** pane to show that Android Studio automatically assigned the textPersonName type. Click the **inputType** field to see the

menu of input types:



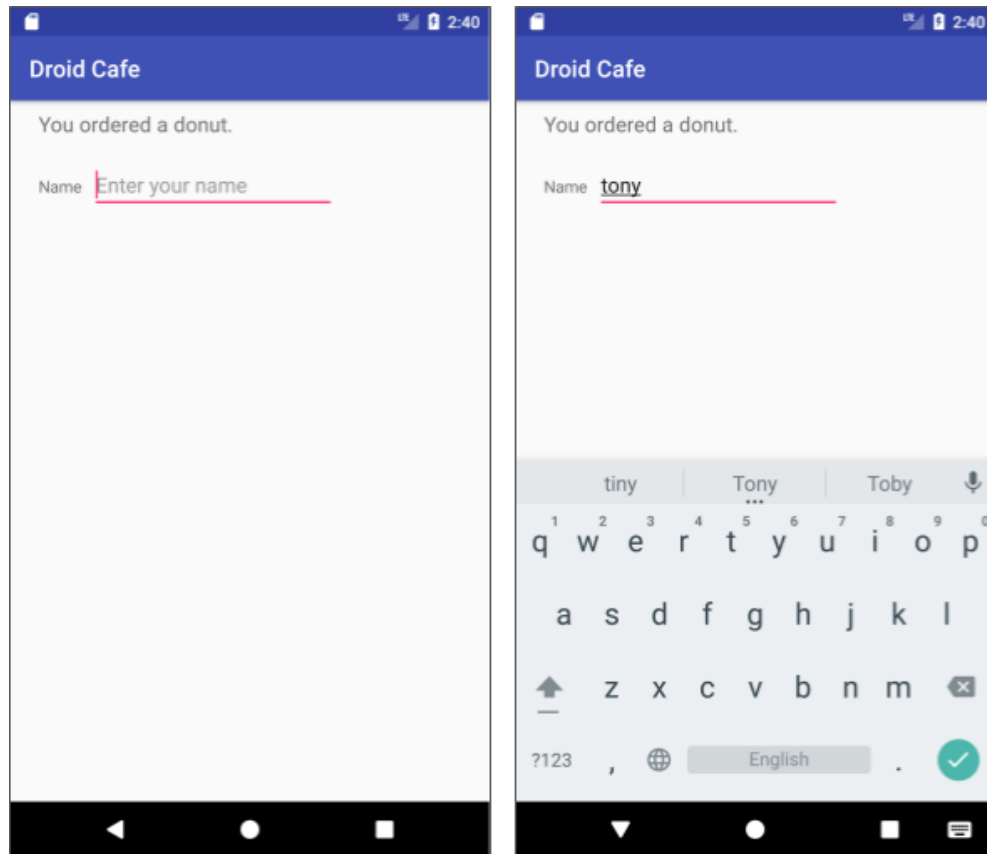
In the figure above, **textPersonName** is selected as the input type.

7. Add a hint for text entry, such as **Enter your name**, in the **hint** field in the **Attributes** pane, and delete the **Name** entry in the **text** field. As a hint to the user, the text "Enter your name" should be dimmed inside the EditText.
8. Check the XML code for the layout by clicking the **Text** tab. Extract the string resource for the `android:hint` attribute value to `enter_name_hint`. The following attributes should be set for the new EditText (add the `layout_marginLeft` attribute for compatibility with older versions of Android):


EditText attribute	Value
android:id	"@+id/name_text"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginStart	8dp
android:layout_marginLeft	8dp
android:ems	"10"
android:hint	"@string/enter_name_hint"
android:inputType	"textPersonName"
app:layout_constraintBaseline_toBaselineOf	"@+id/name_label"
app:layout_constraintStart_toEndOf	"@+id/name_label"

As you can see in the XML code, the [android:inputType](#) attribute is set to textPersonName.

9. Run the app. Tap the donut image on the first screen, and then tap the floating action button to see the next Activity. Tap inside the text entry field to show the keyboard and enter text, as shown in the figure below.



Note that suggestions automatically appear for words that you enter. Tap a suggestion to use it. This is one of the properties of the `textPersonName` value for the [android:inputType](#) attribute. The `inputType` attribute controls a variety of features, including keyboard layout, capitalization, and multiple lines of text.

10. To close the keyboard, tap the checkmark icon in a green circle , which appears in the lower right corner of the keyboard. This is known as the **Done** key.

1.2 Add a multiple-line EditText

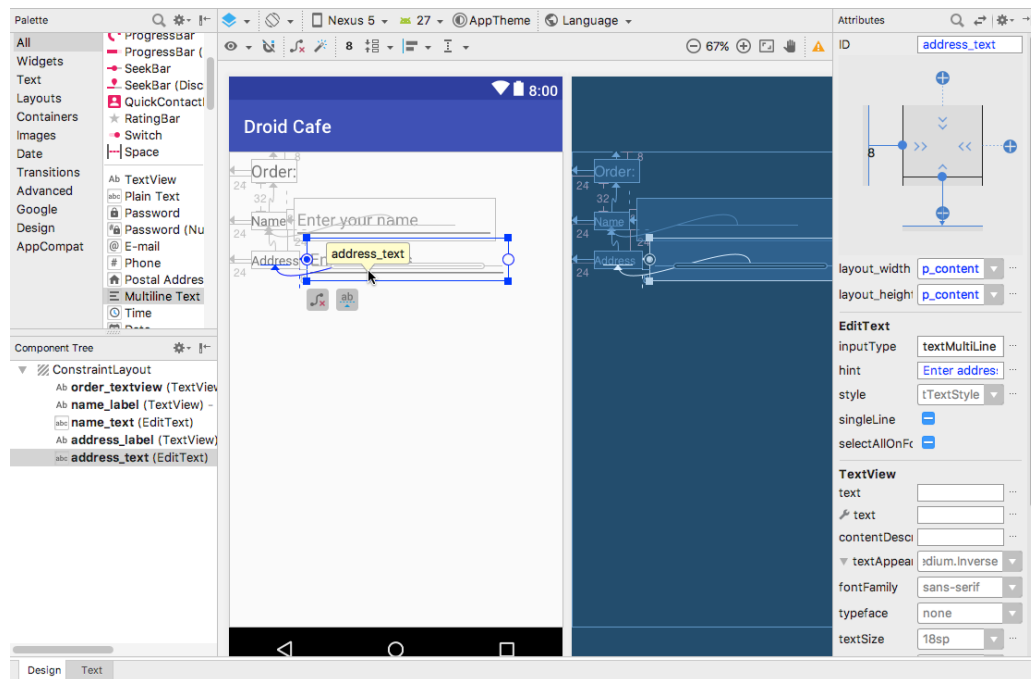
In this step you add another EditText to the OrderActivity layout in the DroidCafe app so that the user can enter an address using multiple lines.

1. Open the **activity_order.xml** layout file if it is not already open.
2. Add a TextView under the name_label element already in the layout. Use the following attributes for the new TextView:

TextView attribute	Value
android:id	"@+id/address_label"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginStart	"24dp"
android:layout_marginLeft	"24dp"
android:layout_marginTop	"24dp"
android:text	"Address"
app:layout_constraintStart_toStartOf	"parent"
app:layout_constraintTop_toBottomOf	"@+id/name_label"

3. Extract the string resource for the android:text attribute value to create an entry for it called address_label_text in strings.xml.
4. Add an EditText element. To use the visual layout editor, drag a **Multiline Text** element from the **Palette** pane to a position next to the address_label TextView. Then enter **address_text** for the **ID** field, and constrain the left side and baseline of the element to the

address_label element right side and baseline as shown in the figure below:



5. Add a hint for text entry, such as **Enter address**, in the **hint** field in the **Attributes** pane. As a hint to the user, the text "Enter address" should be dimmed inside the EditText.
6. Check the XML code for the layout by clicking the **Text** tab. Extract the string resource for the android:hint attribute value to enter_address_hint. The following attributes should be set for the new EditText (add the layout_marginLeft attribute for compatibility with older versions of Android):

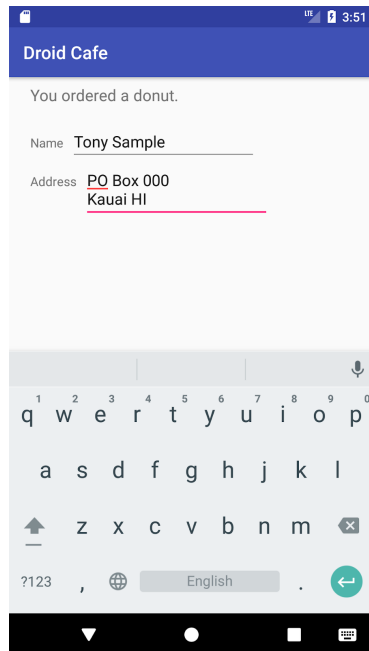
EditText attribute	Value
android:id	"@+id/address_text"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginStart	8dp

android:layout_marginLeft	8dp
android:ems	"10"
android:hint	"@string/enter_address_hint"
android:inputType	"textMultiLine"
app:layout_constraintBaseline_toBaselineOf	"@+id/address_label"
app:layout_constraintStart_toEndOf	"@+id/address_label"

7. Run the app. Tap an image on the first screen, and then tap the floating action button to see the next Activity.
8. Tap inside the "Address" text entry field to show the keyboard and enter text, as shown in



the figure below, using the Return key in the lower right corner of the keyboard (also known as the Enter or New Line key) to start a new line of text. The Return key appears if you set the textMultiLine value for the [android:inputType](#) attribute.



9. To close the keyboard, tap the down-arrow button that appears instead of the Back button in the bottom row of buttons.

1.3 Use a keypad for phone numbers

In this step you add another `EditText` to the `OrderActivity` layout in the `DroidCafe` app so that the user can enter a phone number on a numeric keypad.

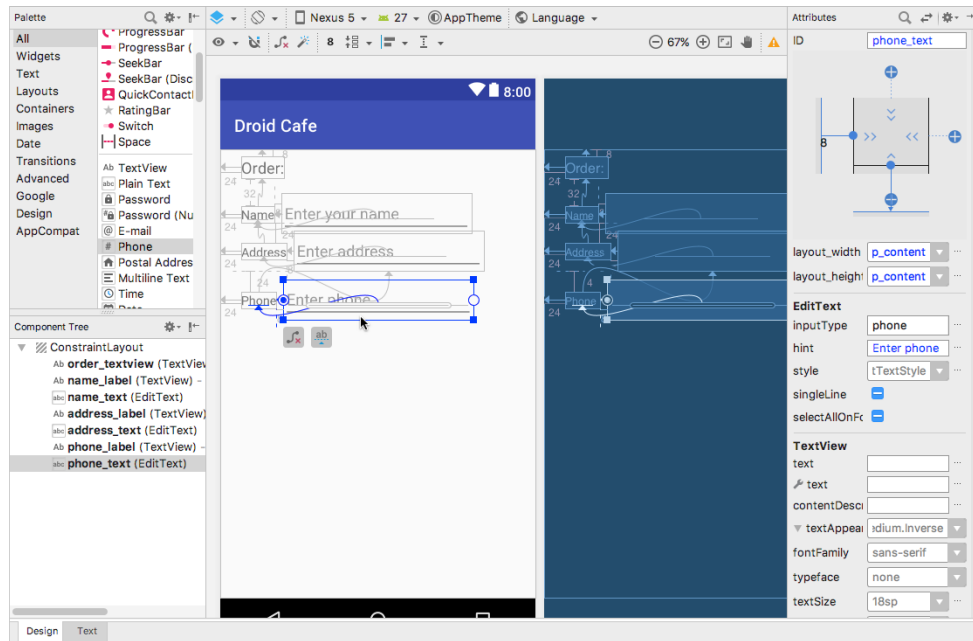
1. Open the **activity_order.xml** layout file if it is not already open.
2. Add a `TextView` under the `address_label` element already in the layout. Use the following attributes for the new `TextView`:

TextView attribute	Value
<code>android:id</code>	<code>"@+id/phone_label"</code>

android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginStart	"24dp"
android:layout_marginLeft	"24dp"
android:layout_marginTop	"24dp"
android:text	"Phone"
app:layout_constraintStart_toStartOf	"parent"
app:layout_constraintTop_toBottomOf	"@+id/address_text"

Note that this TextView is constrained to the bottom of the multiple-line EditText (address_text). This is because address_text can grow to multiple lines, and this TextView should appear beneath it.

3. Extract the string resource for the android:text attribute value to create an entry for it called phone_label_text in strings.xml.
4. Add an EditText element. To use the visual layout editor, drag a **Phone** element from the **Palette** pane to a position next to the phone_label TextView. Then enter **phone_text** for the **ID** field, and constrain the left side and baseline of the element to the phone_label element right side and baseline as shown in the figure below:

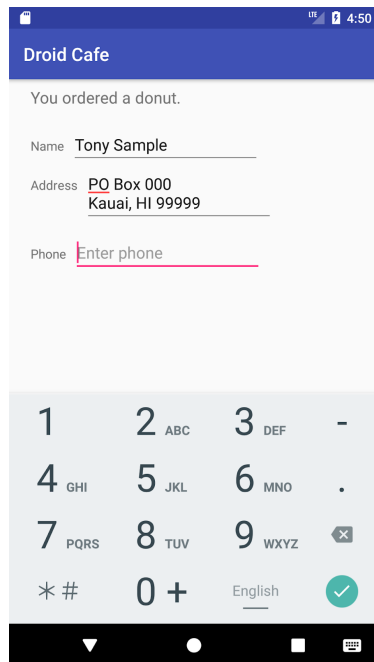



5. Add a hint for text entry, such as **Enter phone**, in the **hint** field in the **Attributes** pane. As a hint to the user, the text "Enter phone" should be dimmed inside the EditText.
6. Check the XML code for the layout by clicking the **Text** tab. Extract the string resource for the `android:hint` attribute value to `enter_phone_hint`. The following attributes should be set for the new EditText (add the `layout_marginLeft` attribute for compatibility with older versions of Android):

EditText attribute	Value
<code>android:id</code>	<code>"@+id/phone_text"</code>
<code>android:layout_width</code>	<code>"wrap_content"</code>
<code>android:layout_height</code>	<code>"wrap_content"</code>
<code>android:layout_marginStart</code>	<code>8dp</code>
<code>android:layout_marginLeft</code>	<code>8dp</code>

android:ems	"10"
android:hint	"@string/enter_phone_hint"
android:inputType	"phone"
app:layout_constraintBaseline_toBaselineOf	"@+id/phone_label"
app:layout_constraintStart_toEndOf	"@+id/phone_label"

7. Run the app. Tap an image on the first screen, and then tap the floating action button to see the next Activity.
8. Tap inside the "Phone" field to show the numeric keypad. You can then enter a phone number, as shown in the figure below.



9. To close the keyboard, tap the **Done** key .

To experiment with `android:inputType` attribute values, change an `EditText` element's `android:inputType` values to the following to see the result:

- `textEmailAddress`: Tapping the field brings up the email keyboard with the @ symbol located near the space key.
- `textPassword`: The characters the user enters turn into dots to conceal the entered password.

1.4 Combine input types in one EditText

You can combine `inputType` attribute values that don't conflict with each other. For example, you can combine the `textMultiLine` and `textCapSentences` attribute values for multiple lines of text in which each sentence starts with a capital letter.

1. Open the **activity_order.xml** layout file if it is not already open.
2. Add a `TextView` under the `phone_label` element already in the layout. Use the following attributes for the new `TextView`:

TextView attribute	Value
<code>android:id</code>	<code>"@+id/note_label"</code>
<code>android:layout_width</code>	<code>"wrap_content"</code>
<code>android:layout_height</code>	<code>"wrap_content"</code>
<code>android:layout_marginStart</code>	<code>"24dp"</code>
<code>android:layout_marginLeft</code>	<code>"24dp"</code>

android:layout_marginTop	"24dp"
android:text	"Note"
app:layout_constraintStart_toStartOf	"parent"
app:layout_constraintTop_toBottomOf	"@+id/phone_label"

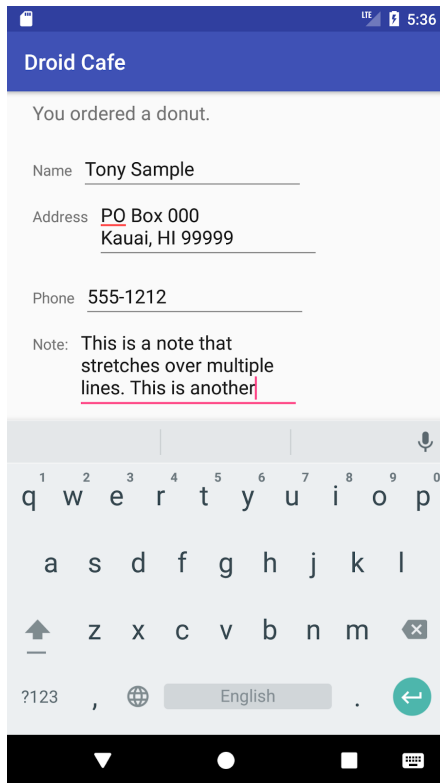
3. Extract the string resource for the android:text attribute value to create an entry for it called note_label_text in strings.xml.
4. Add an EditText element. To use the visual layout editor, drag a **Multiline Text** element from the **Palette** pane to a position next to the note_label TextView. Then enter **note_text** for the **ID** field, and constrain the left side and baseline of the element to the note_label element right side and baseline as you did previously with the other EditText elements.
5. Add a hint for text entry, such as **Enter note**, in the **hint** field in the **Attributes** pane.
6. Click inside the **inputType** field in the **Attributes** pane. The **textMultiLine** value is already selected. In addition, select **textCapSentences** to combine these attributes.
7. Check the XML code for the layout by clicking the **Text** tab. Extract the string resource for the android:hint attribute value to enter_note_hint. The following attributes should be set for the new EditText (add the layout_marginLeft attribute for compatibility with older versions of Android):

EditText attribute	Value
android:id	"@+id/note_text"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginStart	8dp
android:layout_marginLeft	8dp

android:ems	"10"
android:hint	"@string/enter_note_hint"
android:inputType	"textCapSentences textMultiLine"
app:layout_constraintBaseline_toBaselineOf	"@+id/note_label"
app:layout_constraintStart_toEndOf	"@+id/note_label"

To combine values for the `android:inputType` attribute, concatenate them using the pipe (|) character.

8. Run the app. Tap an image on the first screen, and then tap the floating action button to see the next Activity.
9. Tap inside the "Note" field enter complete sentences, as shown in the figure below. Use the Return key to create a new line, or simply type to wrap sentences over multiple lines.



Task 2: Use radio buttons

Input controls are the interactive elements in your app's UI that accept data input. Radio buttons are input controls that are useful for selecting only one option from a set of options.

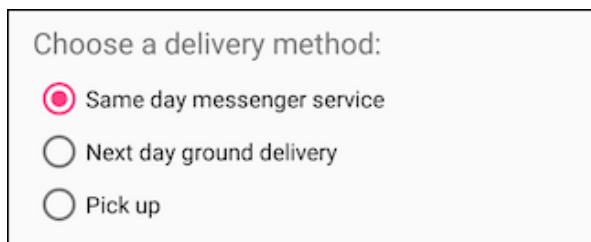
Tip: You should use radio buttons if you want the user to see all available options side-by-side. If it's not necessary to show all options side-by-side, you may want to use a [Spinner](#) instead, which

is described in another chapter.

In this task you add a group of radio buttons to the DroidCafeInput app for setting the delivery options for the dessert order. For an overview and more sample code for radio buttons, see [Radio Buttons](#).

2.1 Add a RadioGroup and radio buttons

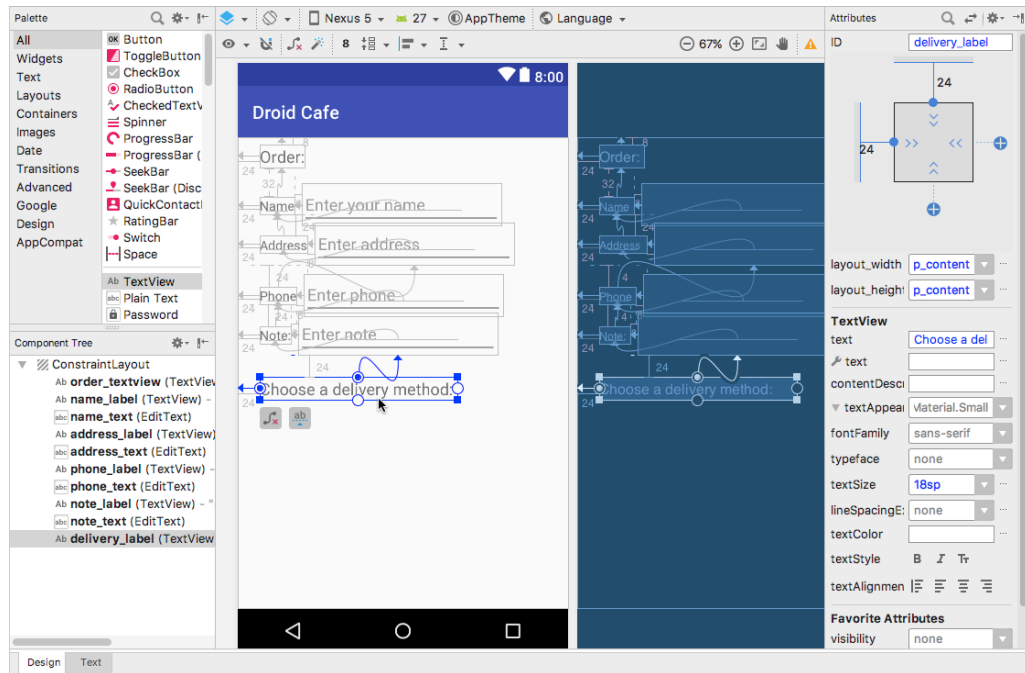
To add radio buttons to OrderActivity in the DroidCafeInput app, you create [RadioButton](#) elements in the activity_order.xml layout file. After editing the layout file, the layout for the radio buttons in OrderActivity will look something like the figure below.



Because radio button selections are mutually exclusive, you group them together inside a [RadioGroup](#). By grouping them together, the Android system ensures that only one radio button can be selected at a time.

Note: The order in which you list the RadioButton elements determines the order that they appear on the screen.

1. Open **activity_order.xml** and add a TextView element constrained to the bottom of the note_text element already in the layout, and to the left margin, as shown in the figure below.



- Switch to editing XML, and make sure that you have the following attributes set for the new TextView:

TextView attribute	Value
android:id	"@+id/delivery_label"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginStart	"24dp"
android:layout_marginLeft	"24dp"
android:layout_marginTop	"24dp"
android:text	"Choose a delivery method: "

android:textSize	"18sp"
app:layout_constraintStart_toStartOf	"parent"
app:layout_constraintTop_toBottomOf	"@+id/note_text"

3. Extract the string resource for "Choose a delivery method:" to be `choose_delivery_method`.
4. To add radio buttons, enclose them within a `RadioGroup`. Add the `RadioGroup` to the layout underneath the `TextView` you just added, enclosing three [RadioButton](#) elements as shown in the XML code below:

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="24dp"
    android:layout_marginStart="24dp"
    android:orientation="vertical"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/delivery_label">

    <RadioButton
        android:id="@+id/sameday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Same day messenger service" />

    <RadioButton
        android:id="@+id/nextday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Next day ground delivery" />

    <RadioButton
        android:id="@+id/pickup"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Pick up" />
</RadioGroup>
```

The "onRadioButtonClicked" entry for the android:onClick attribute for each RadioButton will be underlined in red until you add that method in the next step of this task.

5. Extract the three string resources for the android:text attributes to the following names so that the strings can be translated easily: same_day_messenger_service, next_day_ground_delivery, and pick_up.

2.2 Add the radio button click handler

The android:onClick attribute for each radio button element specifies the onRadioButtonClicked() method to handle the click event. Therefore, you need to add a new onRadioButtonClicked() method in the OrderActivity class.

1. Open **activity_order.xml** (if it is not already open) and find one of the onRadioButtonClicked values for the android:onClick attribute that is underlined in red.
2. Click the onRadioButtonClicked value, and then click the red bulb warning icon in the left margin.
3. Choose **Create onRadioButtonClicked(View) in OrderActivity** in the red bulb's menu. Android Studio creates the onRadioButtonClicked(View view) method in OrderActivity:

```
public void onRadioButtonClicked(View view) {
}
```

In addition, the onRadioButtonClicked values for the other android:onClick attributes in activity_order.xml are resolved and no longer underlined.

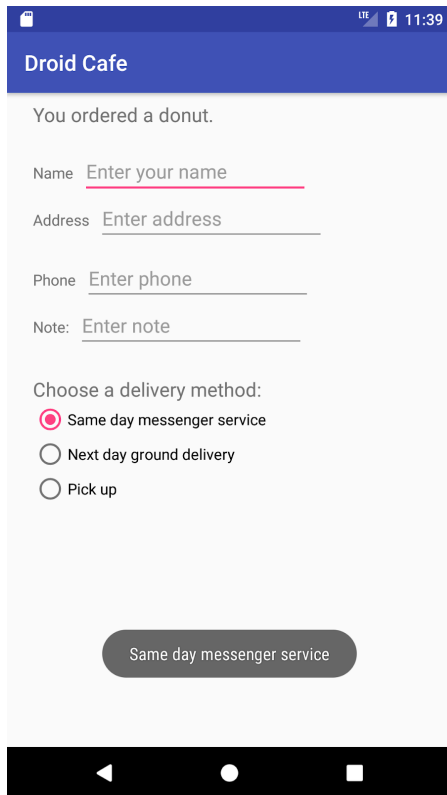
4. To display which radio button is clicked (that is, the type of delivery the user chooses), use a Toast message. Open **OrderActivity** and add the following displayToast method:

```
public void displayToast(String message) {  
    Toast.makeText(getApplicationContext(), message,  
                    Toast.LENGTH_SHORT).show();  
}
```

5. In the new `onRadioButtonClicked()` method, add a switch case block to check which radio button has been selected and to call `displayToast()` with the appropriate message. The code uses the `isChecked()` method of the `Checkable` interface, which returns true if the button is selected. It also uses the `View.getId()` method to get the identifier for the selected radio button view:

```
public void onRadioButtonClicked(View view) {  
    // Is the button now checked?  
    boolean checked = ((RadioButton) view).isChecked();  
    // Check which radio button was clicked.  
    switch (view.getId()) {  
        case R.id.sameday:  
            if (checked)  
                // Same day service  
                displayToast(getString(R.string.same_day_messenger_service));  
            break;  
        case R.id.nextday:  
            if (checked)  
                // Next day delivery  
                displayToast(getString(R.string.next_day_ground_delivery));  
            break;  
        case R.id.pickup:  
            if (checked)  
                // Pick up  
                displayToast(getString(R.string.pick_up));  
            break;  
        default:  
            // Do nothing.  
            break;  
    }  
}
```


6. Run the app. Tap an image to see the `OrderActivity` activity, which shows the delivery choices. Tap a delivery choice, and you see a Toast message at the bottom of the screen with the choice, as shown in the figure below.



Task 2 solution code

Android Studio project: [DroidCafeInput](#)

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: The radio buttons for delivery choices in the DroidCafeInput app first appear unselected, which implies that there is no default delivery choice. Change the radio buttons so that one of them (such as nextday) is selected as the default when the radio buttons first appear.

Hint: You can accomplish this task entirely in the layout file. As an alternative, you can write code in `OrderActivity` to select one of the radio buttons when the Activity first appears.

Challenge solution code

Android Studio project: [DroidCafeInput](#) (see the second radio button in the `activity_order.xml` layout file)

Task 3: Use a spinner for user choices

A [Spinner](#) provides a quick way to select one value from a set. Touching the Spinner displays a drop-down list with all available values, from which the user can select one. If you are providing only two or three choices, you might want to use radio buttons for the choices if you have room in your layout for them; however, with more than three choices, a Spinner works very well, scrolls as needed to display items, and takes up little room in your layout.


Tip: For more information about spinners, see [Spinners](#).

To provide a way to select a label for a phone number (such as **Home**, **Work**, **Mobile**, or **Other**), you can add a spinner to the OrderActivity layout in the DroidCafe app to appear right next to the phone number field.

3.1 Add a spinner to the layout

To add a spinner to the OrderActivity layout in the DroidCafe app, follow these steps, which are numbered in the figure below:

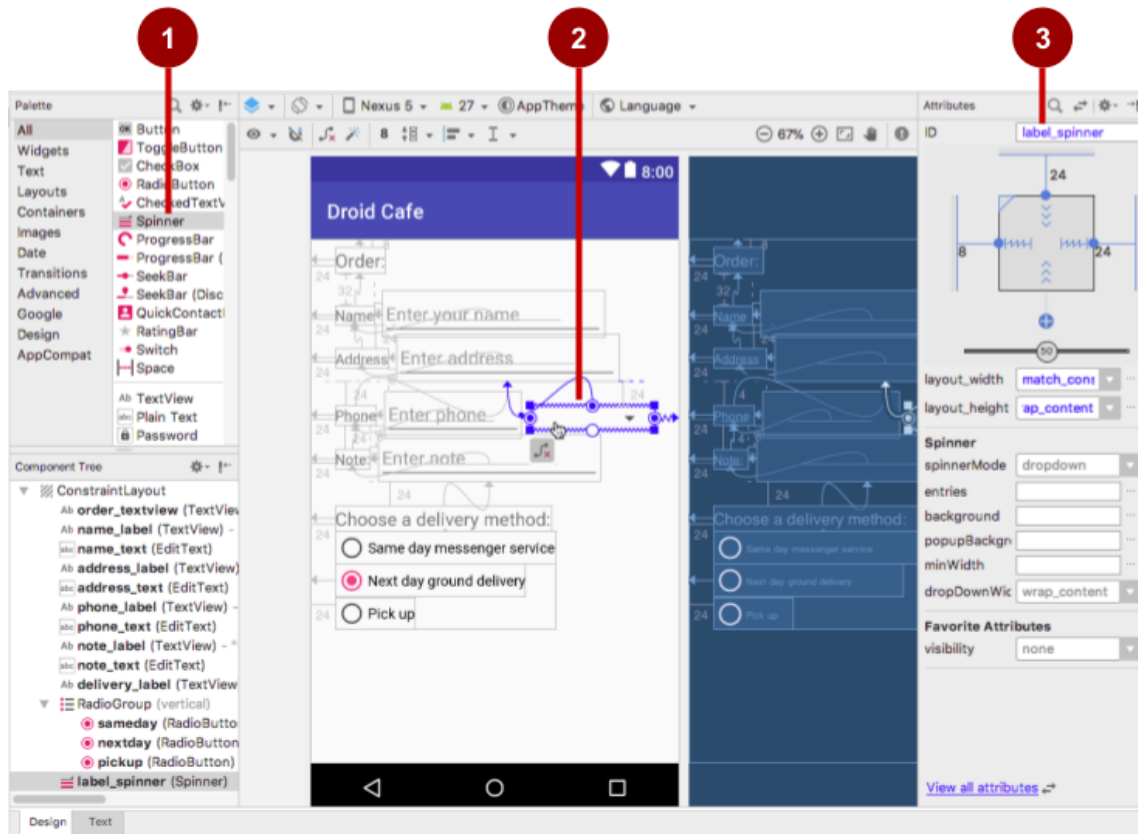
1. Open **activity_order.xml** and drag **Spinner** from the **Palette** pane to the layout.
2. Constrain the top of the Spinner element to the bottom of address_text, the right side to the right side of the layout, and the left side to phone_text.

To align the Spinner and phone_text elements horizontally, use the pack button  in the toolbar, which provides options for packing or expanding selected UI elements.

Select both the Spinner and phone_text elements in the **Component Tree**, click the pack button, and choose **Expand Horizontally**. As a result, both the Spinner and phone_text elements are set to fixed widths.

3. In the Attributes pane, set the Spinner **ID** to **label_spinner**, and set the top and right margins to **24**, and the left margin to **8**. Choose **match_constraint** for the **layout_width** drop-down menu, and **wrap_content** for the **layout_height** drop-down menu.

The layout should look like the figure below. The phone_text element's **layout_width** drop-down menu in the Attributes pane is set to 134dp. You can optionally experiment with other width settings.



To look at the XML code for `activity_order.xml`, click the **Text** tab.

The Spinner should have the following attributes:

```
<Spinner
    android:id="@+id/label_spinner"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="24dp"
    android:layout_marginRight="24dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="24dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/phone_text"
    app:layout_constraintTop_toBottomOf="@+id/address_text" />
```

Be sure to add the `android:layout_marginRight` and `android:layout_marginLeft` attributes shown in the code snippet above to maintain compatibility with older versions of Android.

The `phone_text` element should now have the following attributes (after using the pack tool):

```
<EditText
    android:id="@+id/phone_text"
    android:layout_width="134dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:ems="10"
    android:hint="@string/enter_phone_hint"
    android:inputType="phone"
    app:layout_constraintBaseline_toBaselineOf="@+id/phone_label"
    app:layout_constraintStart_toEndOf="@+id/phone_label" />
```

3.2 Add code to activate the Spinner and its listener

The choices for the [Spinner](#) are well-defined static strings such as "Home" and "Work," so you can use a text array defined in `strings.xml` to hold the values for it.

To activate the Spinner and its listener, implement the [AdapterView.OnItemSelectedListener](#) interface, which requires also adding the `onItemSelected()` and `onNothingSelected()` callback methods.

1. Open **strings.xml** and define the selectable values (**Home**, **Work**, **Mobile**, and **Other**) for the Spinner as the string array `labels_array`:

```
<string-array name="labels_array">

    <item>Home</item>

    <item>Work</item>

    <item>Mobile</item>

    <item>Other</item>

</string-array>
```

2. To define the selection callback for the Spinner, change your OrderActivity class to implement the AdapterView.OnItemClickListener interface as shown:

```
public class OrderActivity extends AppCompatActivity implements
    AdapterView.OnItemClickListener {
```

As you type **AdapterView** in the statement above, Android Studio automatically imports the AdapterView widget. The reason why you need the AdapterView is because you need an adapter—specifically an [ArrayAdapter](#)—to assign the array to the Spinner. An *adapter* connects your data—in this case, the array of spinner items—to the Spinner. You learn more about this pattern of using an adapter to connect data in another practical. This line should appear in your block of import statements:

```
import android.widget.AdapterView;
```

After typing **OnItemSelectedListener** in the statement above, wait a few seconds for a red light bulb to appear in the left margin.

3. Click the light bulb and select **Implement methods**. The `onItemSelected()` and `onNothingSelected()` methods, which are required for `OnItemSelectedListener`, should be highlighted, and the “Insert @Override” option should be selected. Click **OK**.

This step automatically adds empty `onItemSelected()` and `onNothingSelected()` callback methods to the bottom of the `OrderActivity` class. Both methods use the parameter `AdapterView<?>`. The `<?>` is a Java type wildcard, enabling the method to be flexible enough to accept any type of `AdapterView` as an argument.

4. Instantiate a `Spinner` in the `onCreate()` method using the `label_spinner` element in the layout, and set its listener (`spinner.setOnItemSelectedListener()`) in the `onCreate()` method, as shown in the following code snippet:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ... Rest of onCreate code ...

    // Create the spinner.
    Spinner spinner = findViewById(R.id.label_spinner);
    if (spinner != null) {
        spinner.setOnItemSelectedListener(this);
    }

    // Create ArrayAdapter using the string array and default spinner layout.
```

5. Continuing to edit the `onCreate()` method, add a statement that creates the `ArrayAdapter` with the string array (`labels_array`) using the Android-supplied `Spinner` layout for each item (`layout.simple_spinner_item`):

```
// Create ArrayAdapter using the string array and default spinner layout.
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
```

```
R.array.labels_array, android.R.layout.simple_spinner_item);  
// Specify the layout to use when the list of choices appears.
```

The `simple_spinner_item` layout used in this step, and the `simple_spinner_dropdown_item` layout used in the next step, are the default predefined layouts provided by Android in the [R.layout](#) class. You should use these layouts unless you want to define your own layouts for the items in the Spinner and its appearance.

6. Specify the layout for the Spinner choices to be `simple_spinner_dropdown_item`, and then apply the adapter to the Spinner:

```
// Specify the layout to use when the list of choices appears.  
adapter.setDropDownViewResource  
    (android.R.layout.simple_spinner_dropdown_item);  
// Apply the adapter to the spinner.  
if (spinner != null) {  
    spinner.setAdapter(adapter);  
}  
// ... End of onCreate code ...
```


3.3 Add code to respond to Spinner selections

When the user selects an item in the Spinner, the Spinner receives an on-item-selected event. To handle this event, you already implemented the `AdapterView.OnItemSelectedListener` interface in the previous step, adding empty `onItemSelected()` and `onNothingSelected()` callback methods.

In this step you fill in the code for the `onItemSelected()` method to retrieve the selected item in the Spinner, using `getItemAtPosition()`, and assign the item to the `spinnerLabel` variable:

1. Add code to the empty `onItemSelected()` callback method, as shown below, to retrieve the user's selected item using `getItemAtPosition()`, and assign it to `spinnerLabel`. You can also add a call to the `displayToast()` method you already added to `OrderActivity`:

```
public void onItemSelected(AdapterView<?> adapterView, View view, int
    i, long l) {
    spinnerLabel = adapterView.getItemAtPosition(i).toString();
    displayToast(spinnerLabel);
}
```

There is no need to add code to the empty `onNothingSelected()` callback method for this example.

2. Run the app.

The Spinner appears next to the phone entry field and shows the first choice (**Home**). Tapping the Spinner reveals all the choices, as shown on the left side of the figure below. Tapping a choice in the Spinner shows a Toast message with the choice, as shown on the right side of the figure.

