Practical 6: Clickable images

The user interface (UI) that appears on a screen of an Android-powered device consists of a **hierarchy of objects** called ***views***. Every element of the screen is a **View**.

The **View class** represents the basic building block for all UI components. **View is the base class** for classes that provide interactive UI components, such as **Button elements**. A Button is a UI element the user can tap or click to perform an action.

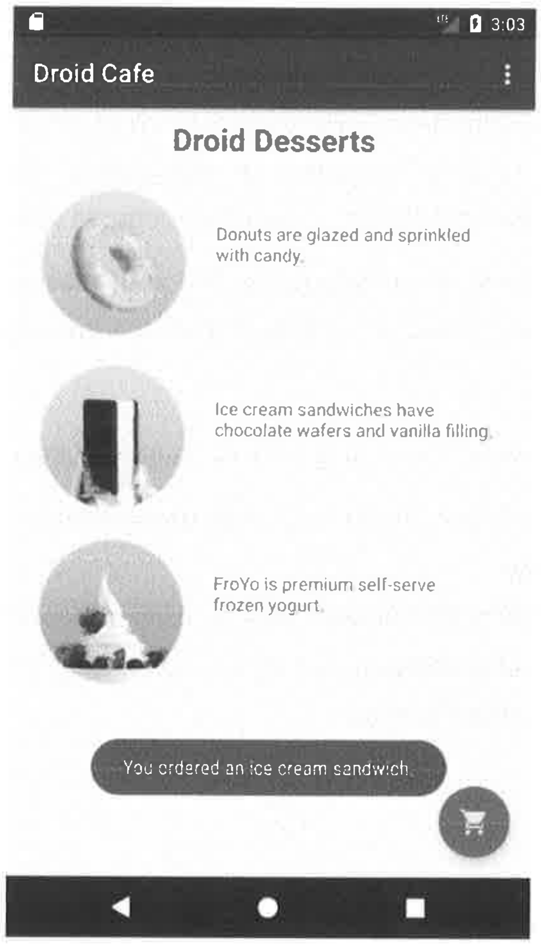
You can turn any View, such as an **ImageView**, into a UI element that can be **tapped** or **clicked**. You must store the image for the **ImageView** in the **drawables folder** of your project.

In this practical, you learn how to use images as elements that the user can tap or click.

- Create a new Android Studio project for a mock **dessert-ordering app** that uses images as interactive elements.
- Set **onClick() handlers** for the images to display different **Toast messages**.
- Change the floating action **button** supplied by the template so that it shows a different icon and **launches another Activity**.

App overview

In this practical, you create and build a new app starting with the Basic Activity template that imitates a dessert-ordering app. The user can tap an image to perform an action—in this case display a Toast message—as shown in the figure below. The user can also tap a shopping-cart button to proceed to the next Activity.



Donuts are glazed and sprinkled with candy.

Ice cream sandwiches have chocolate wafers and vanilla filling.

FroYo is premium self-serve frozen yogurt.

You ordered an ice cream sandwich.

Task 1: Add images to the layout

You can make **a view clickable**, as a button, by adding the **android:onClick attribute** in the XML layout. For example, you can make an **image act like a button** by adding **android:onClick** to the ImageView.

In this task you create a prototype of an app for ordering desserts from a café. After starting a new project based on the Basic Activity template, you modify the "Hello World" TextView with appropriate text, and add images that the user can tap.

1.1 Start the new project

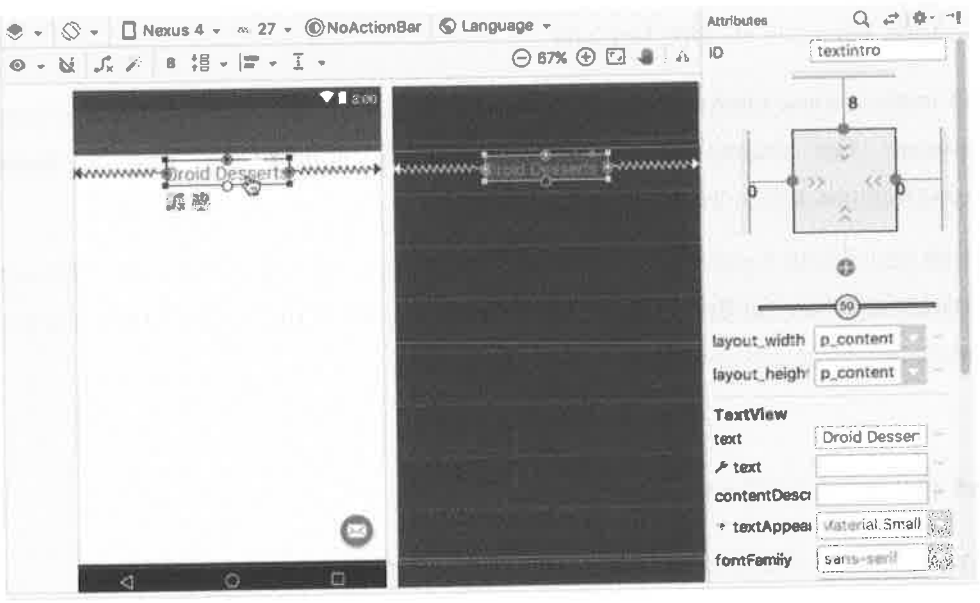
1. Start a new Android Studio project with the app name **Droid Cafe**.
2. Choose the **Basic Activity** template, and accept the default Activity name (MainActivity). Make sure the **Generate Layout file** and **Backwards Compatibility (AppCompat)** options are selected.
3. Click **Finish**.

   The project opens with two layouts in the **res > layout** folder: **activity_main.xml** for the app bar and floating action button (which you don't change in this task), and **content_main.xml** for everything else in the layout.
4. Open fragment_first.xml ~~**content_main.xml**~~ and click the **Design** tab (if it is not already selected) to show the layout editor.
5. Select the "Hello World" TextView in the layout and open the **Attributes** pane.
6. Change the textintro attributes as follows:

Attribute field	**Enter the following:**
ID	textintro
text	Change Hello World to Droid Desserts
textStyle	B (bold)
textSize	24sp

This adds the android:id attribute to the TextView with the id set to textintro, changes the text, makes the text bold, and sets a larger text size of 24sp.

7. Delete the constraint that stretches from the bottom of the textintro TextView to the bottom of the layout, so that the TextView snaps to the top of the layout, and choose **8** (8dp) for the top margin as shown below.



ID: textintro
layout_width: p_content
layout_height: p_content
text: Droid Desser
contentDescription:
textAppearance: Material.Small
fontFamily: sans-serif

8. In a previous lesson you learned how to extract a string resource from a literal text string. Click the **Text** tab to switch to XML code, and extract the "Droid Desserts" string in the TextView and enter **intro_text** as the string resource name.

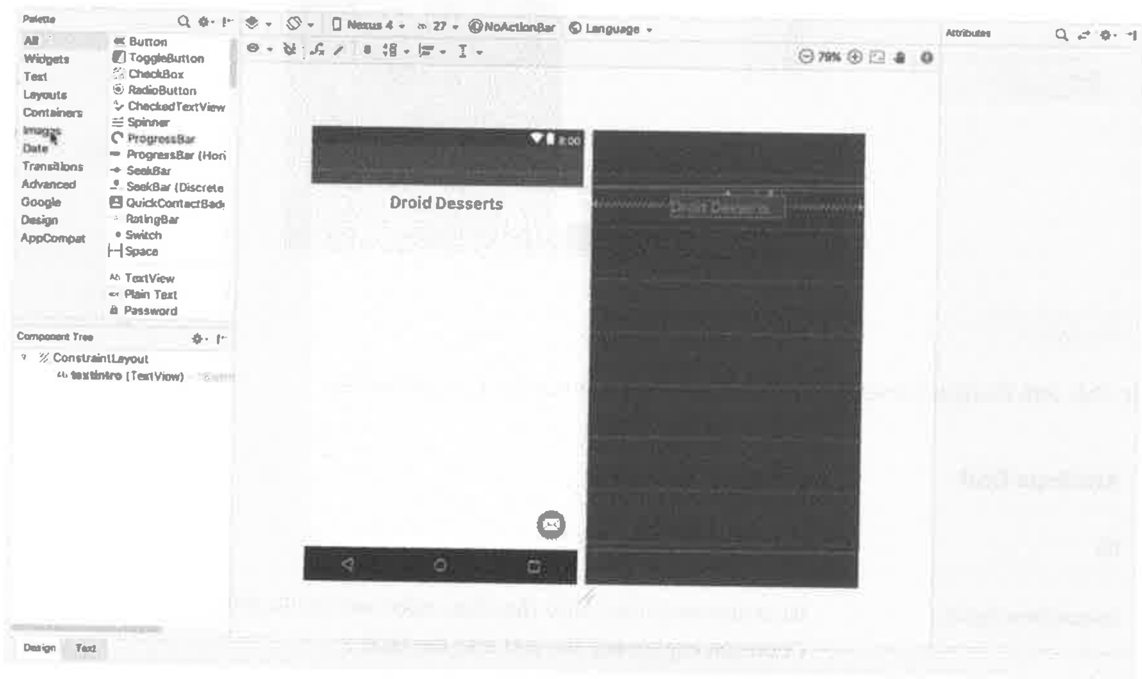
1.2 Add the images

Three images (donut_circle.png, froyo_circle.png, and icecream_circle.png) are provided for this example, which you can download. As an alternative, you can substitute your own images as PNG files, but they must be sized at about 113 x 113 pixels to use in this example.

This step also introduces a new technique in the layout editor: using the **Fix** button in warning messages to extract string resources.

1. To copy the images to your project, first close the project.
2. Copy the image files into your project's **drawable** folder. Find the **drawable** folder in a project by using this path: *project_name* **> app > src > main > res > drawable**.
3. Reopen your project.
4. Open **content_main.xml** file, and click the **Design** tab (if it is not already selected).

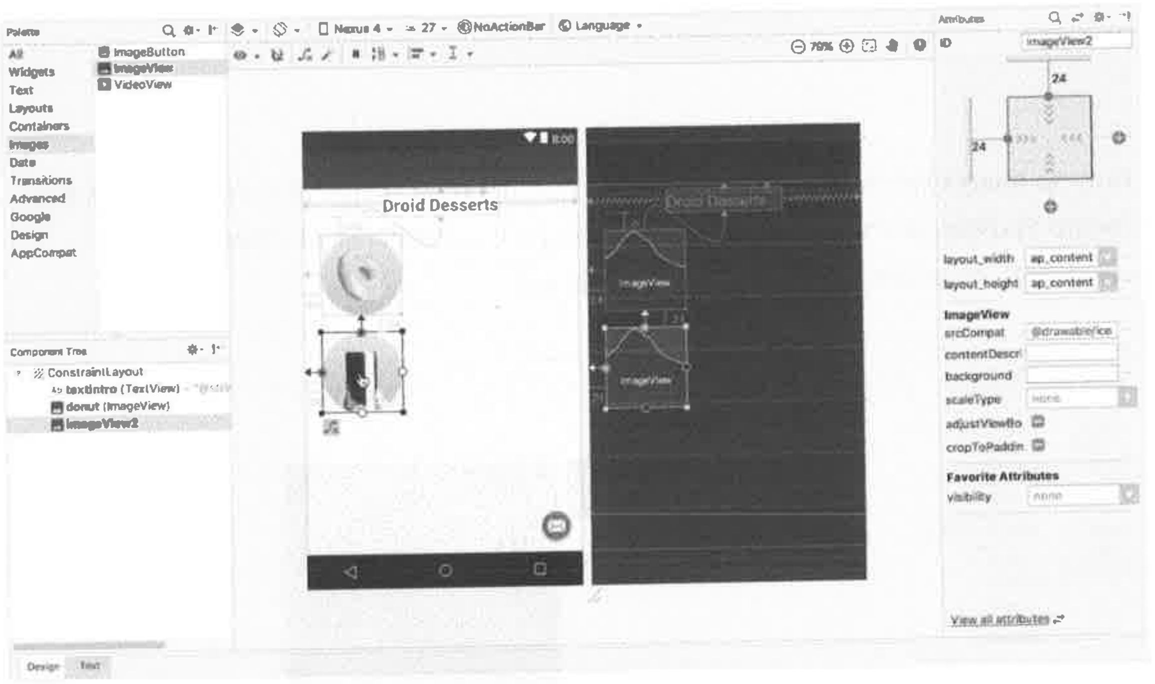
5. Drag an **ImageView** to the layout, choose the **donut_circle** image for it, and constrain it to the top TextView and to the left side of the layout with a margin of **24** (24dp) for both constraints, as shown in the animated figure below.



6. In the **Attributes** pane, enter the following values for the attributes:

Attribute field	Enter the following:
ID	donut
contentDescription	Donuts are glazed and sprinkled with candy. (You can copy/paste the text into the field.)

7. Drag a second ImageView to the layout, choose the **icecream_circle** image for it, and constrain it to the bottom of the first ImageView and to the left side of the layout with a margin of **24** (24dp) for both constraints.



Attributes

ID imageView2

layout_width wrap_content

layout_height wrap_content

ImageView

srcCompat @drawable/ice

contentDescr

background

scaleType none

adjustViewBo

cropToPaddin

Favorite Attributes

visibility none

Component Tree

ConstraintLayout

textIntro (TextView)

donut (ImageView)

imageView2

8. In the **Attributes** pane, enter the following values for the attributes:

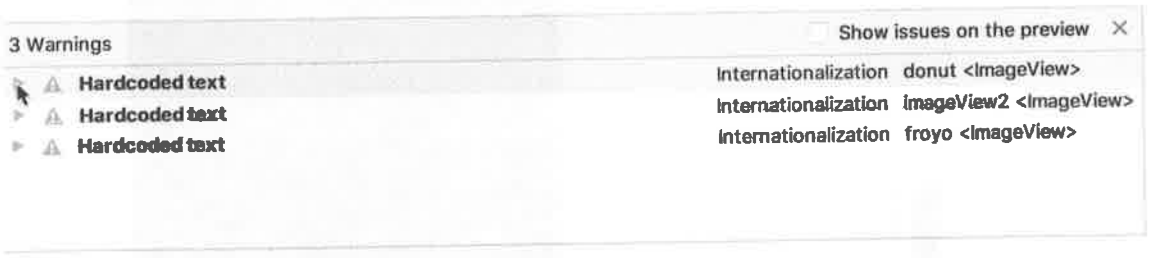
Attribute field	Enter the following:
ID	ice_cream
contentDescription	Ice cream sandwiches have chocolate wafers and vanilla filling. (You can copy/paste the text into the field.)

9. Drag a third ImageView to the layout, choose the **froyo_circle** image for it, and constrain it to the bottom of the second ImageView and to the left side of the layout with a margin of **24** (24dp) for both constraints.

10. In the **Attributes** pane, enter the following values for the attributes:

Attribute field	Enter the following:
ID	froyo
contentDescription	FroYo is premium self-serve frozen yogurt. (You can copy/paste the text into the field.)

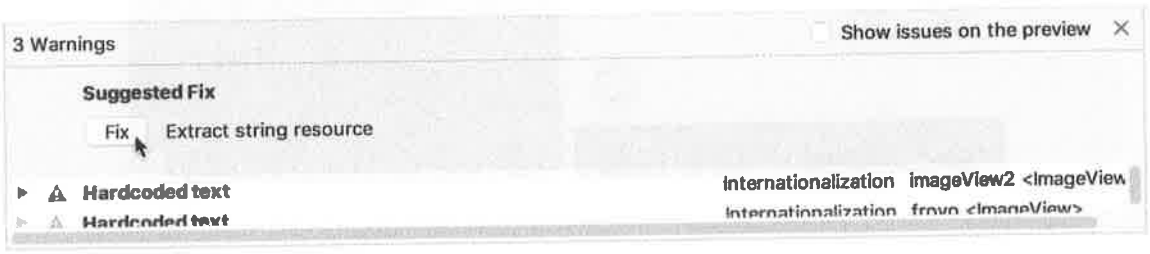
11. Click the warning icon ⚠ in the upper left corner of the layout editor to open the warning pane, which should display warnings about hardcoded text:


3 Warnings

Show issues on the preview ×

▸ ⚠ Hardcoded text	Internationalization	donut <ImageView>
▸ ⚠ Hardcoded text	Internationalization	imageView2 <ImageView>
▸ ⚠ Hardcoded text	Internationalization	froyo <ImageView>

12. Expand each **Hardcoded text** warning, scroll to the bottom of the warning message, and click the **Fix** button as shown below:


3 Warnings

Show issues on the preview ×

Suggested Fix

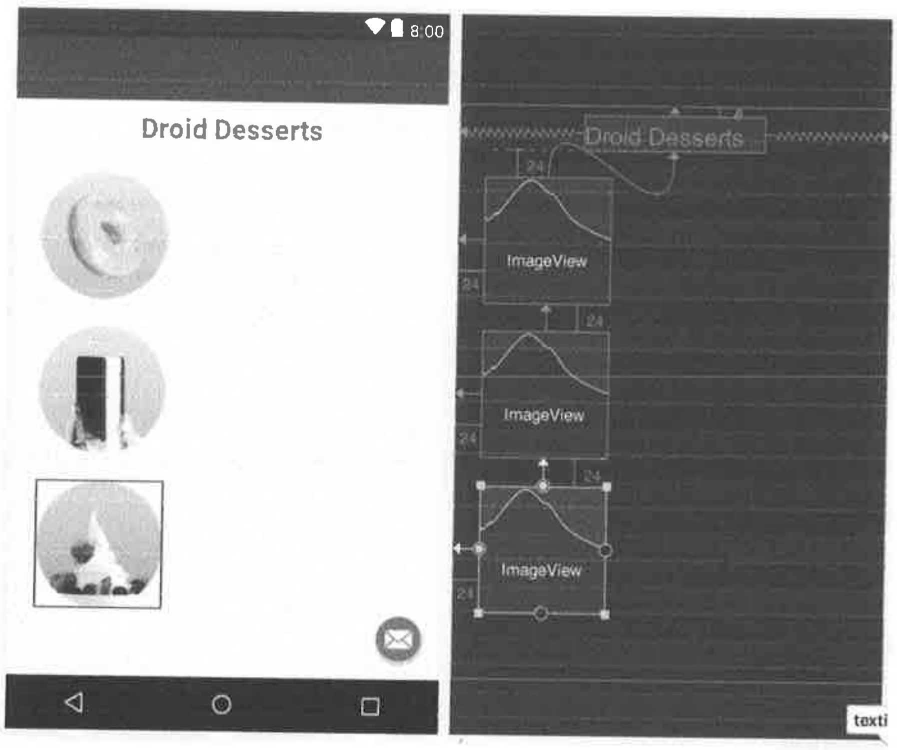
Fix Extract string resource

▸ ⚠ Hardcoded text	Internationalization	imageView2 <ImageView
▸ ⚠ Hardcoded text	Internationalization	froyo <ImageView>

The fix for each hardcoded text warning extracts the string resource for the string. The **Extract Resource** dialog appears, and you can enter the name for the string resource. Enter the following names for the string resources:

String	**Enter the following name:**
Donuts are glazed and sprinkled with candy.	donuts
Ice cream sandwiches have chocolate wafers and vanilla filling.	ice_cream_sandwiches
FroYo is premium self-serve frozen yogurt.	froyo

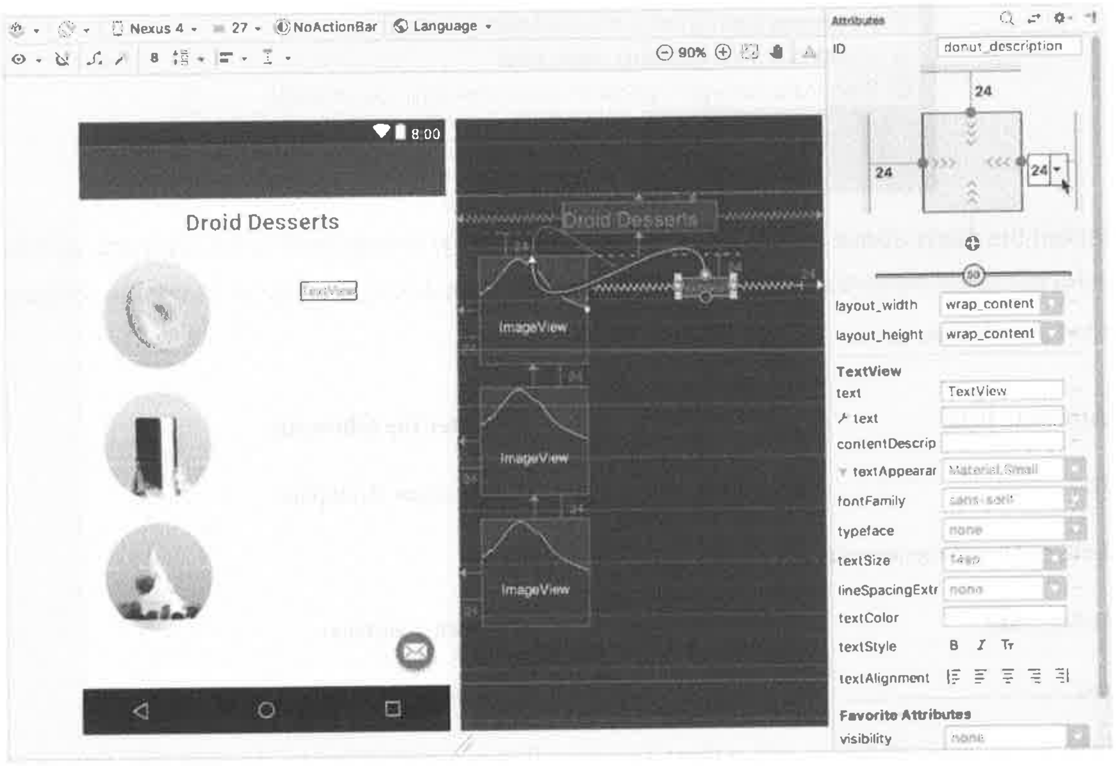
The layout should now look like the figure below.



1.3 Add the text descriptions

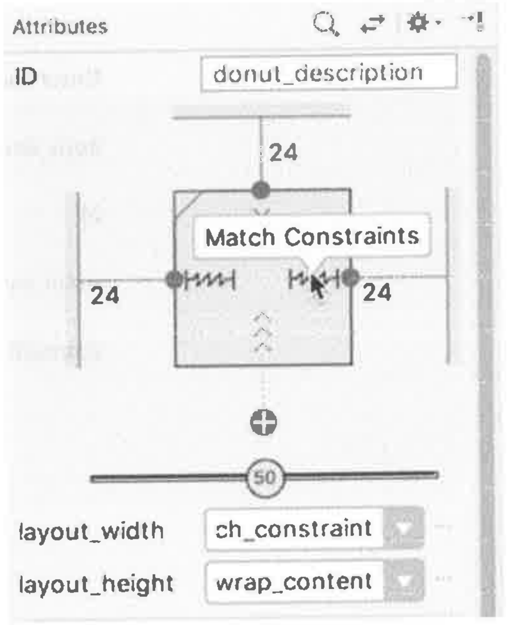
In this step you **add a text description** (TextView) for each dessert. Because you have already extracted string resources for the contentDescription fields for the ImageView elements, you can use the same string resources for each description TextView.

1. Drag a TextView element to the layout.
2. Constrain the element's left side to the right side of the donut ImageView and its top to the top of the donut ImageView, both with a margin of **24** (24dp).
3. Constrain the element's right side to the right side of the layout, and use the same margin of **24** (24dp). Enter **donut_description** for the ID field in the **Attributes** pane. The new TextView should appear next to the donut image as shown in the figure below.



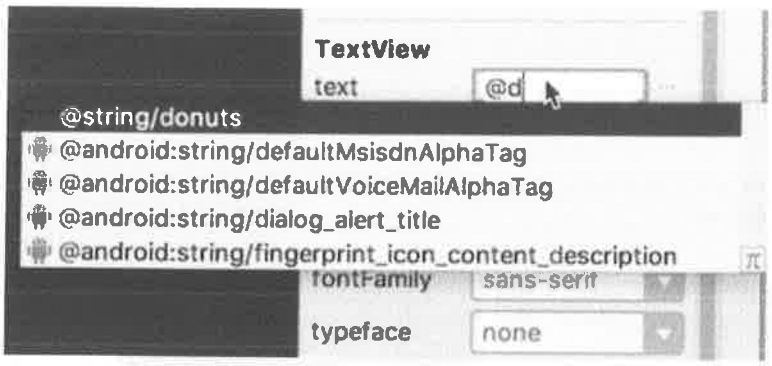
ID: donut_description
layout_width: wrap_content
layout_height: wrap_content
TextView
text: TextView
text
contentDescrip
textAppearar: Material.Small
fontFamily: sans-serif
typeface: none
textSize: 14sp
lineSpacingExtr: none
textColor
textStyle: B I Tr
textAlignment
Favorite Attributes
visibility: none

4. In the **Attributes** pane change the width in the inspector pane to **Match Constraints**:



ID: donut_description
layout_width: ch_constraint
layout_height: wrap_content

5. In the **Attributes** pane, begin entering the string resource for the text field by prefacing it with the @ symbol: **@d**. Click the string resource name (**@string/donuts**) which appears as a suggestion:


TextView
text
@d
@string/donuts
@android:string/defaultMsisdnAlphaTag
@android:string/defaultVoiceMailAlphaTag
@android:string/dialog_alert_title
@android:string/fingerprint_icon_content_description
fontFamily
sans-serif
typeface
none

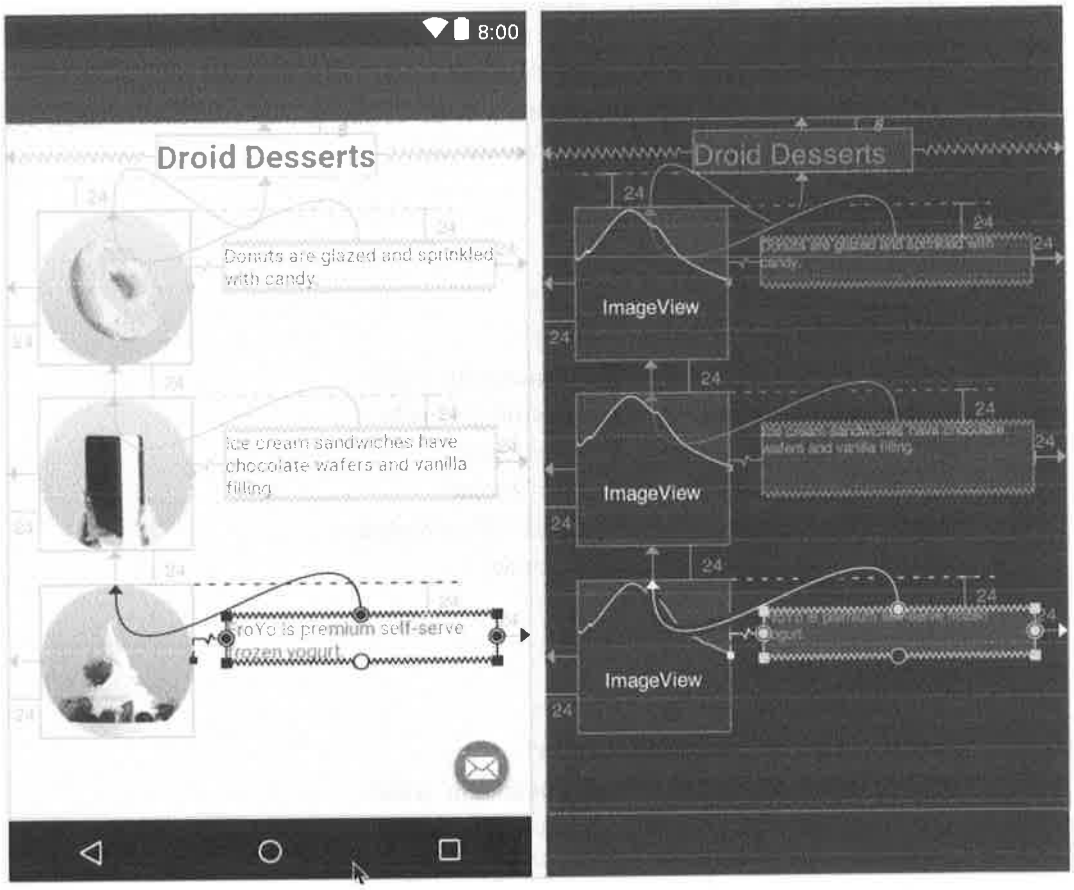
6. Repeat the steps above to add a second TextView that is constrained to the right side and top of the ice_cream ImageView, and its right side to the right side of the layout. Enter the following in the **Attributes** pane:

Attribute field	**Enter the following:**
ID	ice_cream_description
Left, right, and top margins	24
layout_width	match_constraint
text	@string/ice_cream_sandwiches

7. Repeat the steps above to add a third TextView that is constrained to the right side and top of the froyo ImageView, and its right side to the right side of the layout. Enter the following in the **Attributes** pane:

Attribute field	**Enter the following:**
ID	froyo_description
Left, right, and top margins	24
layout_width	match_constraint
text	@string/froyo

The layout should now look like the following:


8:00
Droid Desserts
Donuts are glazed and sprinkled with candy.
Ice cream sandwiches have chocolate wafers and vanilla filling
FroYo is premium self-serve frozen yogurt.
ImageView
ImageView
ImageView

Task 1 solution code

The XML layout for the **content.xml** file is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.example.android.droidcafe.MainActivity"
    tools:showIn="@layout/activity_main">
```

```
<TextView
    android:id="@+id/textintro"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/margin_regular"
    android:text="@string/intro_text"
    android:textSize="@dimen/text_heading"
    android:textStyle="bold"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<ImageView
    android:id="@+id/donut"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:contentDescription="@string/donuts"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textintro"
    app:srcCompat="@drawable/donut_circle" />

<ImageView
    android:id="@+id/ice_cream"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:contentDescription="@string/ice_cream_sandwiches"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/donut"
    app:srcCompat="@drawable/icecream_circle" />

<ImageView
    android:id="@+id/froyo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:contentDescription="@string/froyo"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/ice_cream"
    app:srcCompat="@drawable/froyo_circle" />
```

```
<TextView
    android:id="@+id/donut_description"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="@dimen/margin_wide"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:text="@string/donuts"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/donut"
    app:layout_constraintTop_toTopOf="@+id/donut" />

<TextView
    android:id="@+id/ice_cream_description"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="@dimen/margin_wide"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:text="@string/ice_cream_sandwiches"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/ice_cream"
    app:layout_constraintTop_toTopOf="@+id/ice_cream" />

<TextView
    android:id="@+id/froyo_description"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="@dimen/margin_wide"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:text="@string/froyo"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/froyo"
    app:layout_constraintTop_toTopOf="@+id/froyo" />

</android.support.constraint.ConstraintLayout>
```

Task 2: Add onClick methods for images

To make a View *clickable* so that users can tap (or click) it, add the android:onClick attribute in the XML layout and specify the click handler. For example, you can make an ImageView act like a simple Button by adding android:onClick to the ImageView. In this task you make the images in your layout clickable.

2.1 Create a Toast method

In this task you add each method for the **android:onClick** attribute to call when each image is clicked. In this task, these methods simply **display a Toast message** showing which image was tapped. (In another chapter you modify these methods to launch another Activity.)

1. To use **string resources** in Java code, you should first add them to the **strings.xml file**. Expand **res > values** in the **Project > Android** pane, and open **strings.xml**. Add the following string resources for the strings to be shown in the Toast message:

```
<string name="donut_order_message">You ordered a donut.</string>
<string name="ice_cream_order_message">You ordered an ice cream sandwich.</string>
<string name="froyo_order_message">You ordered a FroYo.</string>
```

2. Open **MainActivity**, and **add** the following **displayToast() method** to the end of **MainActivity** (before the closing bracket):

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

Although you could have added this method in any position within **MainActivity**, it is best practice to put your own methods *below* the methods already provided in **MainActivity** by the template.

2.2 Create click handlers

Each clickable image needs a click handler—a method for the **android:onClick attribute** to call. The click handler, if called from the android:onClick attribute, must be **public**, return **void**, and define a **View** as its only parameter. Follow these steps to add the click handlers:

1. Add the following **showDonutOrder() method** to **MainActivity**. For this task, use the previously created **displayToast() method** to display a Toast message:

```
/**
* Shows a message that the donut image was clicked.
*/
public void showDonutOrder(View view) {
   displayToast(getString(R.string.donut_order_message));
}
```

   The first three lines are a comment in the Javadoc format, which makes the code easier to understand and also helps generate documentation for your code. It is a best practice to add such a comment to every new method you create. For more information about how to write comments, see How to Write Doc Comments for the Javadoc Tool.

2. **Add more methods** to the end of **MainActivity** for each dessert:

```
/**
* Shows a message that the ice cream sandwich image was clicked.
*/
public void showIceCreamOrder(View view) {
   displayToast(getString(R.string.ice_cream_order_message));
}

/**
* Shows a message that the froyo image was clicked.
*/
public void showFroyoOrder(View view) {
   displayToast(getString(R.string.froyo_order_message));
}
```

3. (Optional) Choose **Code > Reformat Code** to reformat the code you added in MainActivity to conform to standards and make it easier to read.

2.3 Add the onClick attribute

In this step you **add android:onClick to each of the ImageView elements** in the **content_main.xml** layout. The android:onClick attribute calls the click handler for each element.

1. Open the **content_main.xml** file, and click the **Text** tab in the layout editor to show the XML code.
2. Add the **android:onClick attribute to donut ImageView**. As you enter it, suggestions appear showing the click handlers. Select the showDonutOrder click handler. The code should now look as follows:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:id="@+id/donut"
    android:layout_below="@id/choose_dessert"
    android:contentDescription="@string/donut"
    android:src="@drawable/donut_circle"
    android:onClick="showDonutOrder"/>
```

The last line (android:onClick="showDonutOrder") assigns the click handler (showDonutOrder) to the ImageView.

3. (Optional) Choose **Code > Reformat Code** to reformat the XML code you added in content_main.xml to conform to standards and make it easier to read. Android Studio automatically moves the android:onClick attribute up a few lines to combine them with the other attributes that have android: as the preface.
4. Follow the same procedure to add the android:onClick attribute to the ice_cream and froyo ImageView elements. Select the showDonutOrder and showFroyoOrder click handlers. You can optionally choose **Code > Reformat Code** to reformat the XML code. The code should now look as follows:

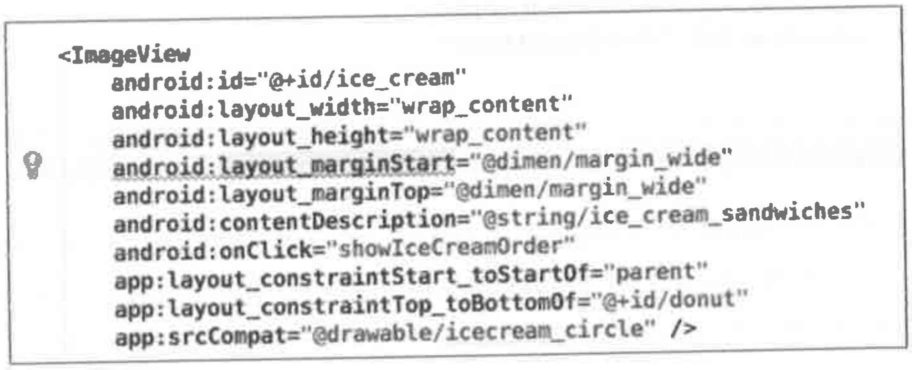
```
<ImageView
    android:id="@+id/ice_cream"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:contentDescription="@string/ice_cream_sandwiches"
    android:onClick="showIceCreamOrder"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/donut"
        app:srcCompat="@drawable/icecream_circle" />

    <ImageView
        android:id="@+id/froyo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="@dimen/margin_wide"
        android:layout_marginTop="@dimen/margin_wide"
        android:contentDescription="@string/froyo"
        android:onClick="showFroyoOrder"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/ice_cream"
        app:srcCompat="@drawable/froyo_circle" />
```

Note that the attribute **android:layout_marginStart** in each ImageView is underlined in **red**. This attribute determines the "start" margin for the ImageView, which is on the left side for most languages but on the right side for languages that read right-to-left (RTL).

5. Click the **android:** preface part of the **android:layout_marginStart** attribute, and a red bulb warning appears next to it, as shown in the figure below.

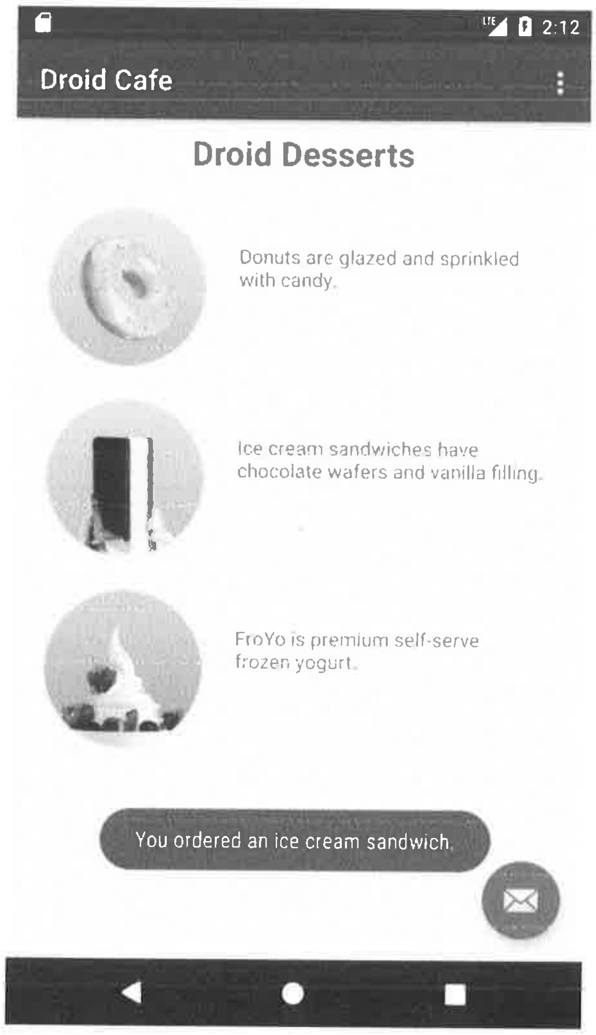


```
<ImageView
    android:id="@+id/ice_cream"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:contentDescription="@string/ice_cream_sandwiches"
    android:onClick="showIceCreamOrder"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/donut"
    app:srcCompat="@drawable/icecream_circle" />
```

6. To make your app compatible with previous versions of Android, **click the red bulb** for each instance of this attribute, and choose **Set layout_marginLeft...** to set the layout_marginLeft to "@dimen/margin_wide".

7. Run the app.

   Clicking the donut, ice cream sandwich, or froyo image displays a Toast message about the order, as shown in the figure below.



Donuts are glazed and sprinkled with candy.

Ice cream sandwiches have chocolate wafers and vanilla filling.

FroYo is premium self-serve frozen yogurt.

You ordered an ice cream sandwich.

Task 2 solution code

The solution code for this task is included in the code and layout for MainActivity in the Android Studio project DroidCafe.

Task 3: Change the floating action button

When you click the floating action button with the email icon that appears at the bottom of the screen, the code in MainActivity displays a brief message in a drawer that opens from the bottom of the screen on a smartphone, or from the lower left corner on larger devices, and then closes after a few seconds. This is called a *snackbar*. It is used to provide feedback about an operation. For more information, see Snackbar.

Look at how other apps implement the floating action button. For example, the Gmail app provides a floating action button to create a new email message, and the Contacts app provides one to create a new contact. For more information about floating action buttons, see FloatingActionButton.

For this task you change the icon for the FloatingActionButton to a shopping cart, and change the action for the FloatingActionButton to launch a new Activity.

3.1 Add a new icon

As you learned in another lesson, you can **choose an icon from the set of icons** in Android Studio. Follow these steps:

1. Expand **res** in the **Project > Android** pane, and right-click (or Control-click) the **drawable** folder.
2. Choose **New > Image Asset**. The Configure Image Asset dialog appears.
3. Choose **Action Bar and Tab Icons** in the drop-down menu at the top of the dialog. (Note that the *action bar* is the same thing as the *app bar*.)
4. Change **ic_action_name** in the **Name** field to **ic_shopping_cart**.
5. Click the clip art image (the Android logo next to **Clipart:**) to select a clip art image as the icon. A page of icons appears. Click the icon you want to use for the floating action button, such as the shopping cart icon.
6. Choose **HOLO_DARK** from the **Theme** drop-down menu. This sets the icon to be white against a dark-colored (or black) background. Click **Next**.
7. Click **Finish** in the Confirm Icon Path dialog.

Tip: For a complete description for adding an icon, see Create app icons with Image Asset Studio.

3.2 Add an Activity

As you learned in a previous lesson, **an Activity represents a single screen** in your app in which your user can perform a single, focused task. You already have one activity, **MainActivity.java.** Now you **add another activity** called **OrderActivity.java.**

1. Right-click (or Control-click) the **com.example.android.droidcafe** folder in the left column and choose **New > Activity > Empty Activity**.
2. Edit the **Activity Name** to be **OrderActivity**, and the **Layout Name** to be **activity_order**. Leave the other options alone, and click **Finish**.

   The OrderActivity class should now be listed along with MainActivity in the **java** folder, and activity_order.xml should now be listed in the **layout** folder. The Empty Activity template added these files.

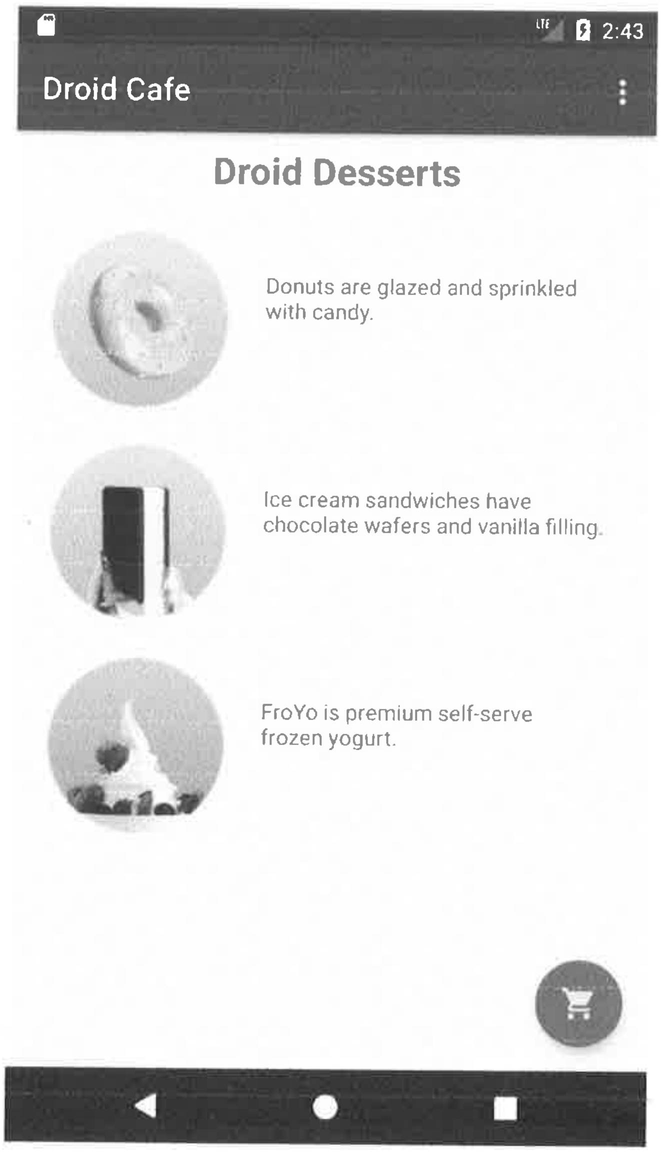
3.3 Change the action

In this step you **change the action** for the **FloatingActionButton** to **launch the new Activity**.

1. Open **MainActivity**.
2. Change the **onClick(View view)** method to make an **explicit intent** to start **OrderActivity**:

```
public void onClick(View view) {
    Intent intent = new Intent(MainActivity.this, OrderActivity.class);
    startActivity(intent);
}
```

3. **Run** the app. Tap the floating action button that now uses the shopping cart icon. A blank Activity should appear (OrderActivity). Tap the Back button to go back to MainActivity.


2:43

Droid Cafe

Droid Desserts

Donuts are glazed and sprinkled with candy.

Ice cream sandwiches have chocolate wafers and vanilla filling.

FroYo is premium self-serve frozen yogurt.

Task 3 solution code

The solution code for this task is included in the code and layout for Android Studio project DroidCafe.