**Quiz: Adding A CheckBox**

[**Link to Common Android Views cheat sheet**](https://drive.google.com/a/udacity.com/file/d/0B5XIkMkayHgRMVljUVIyZzNmQUU/view)

**21\_Quiz: Styles**

For more on styles, check out [**the documentation**](http://developer.android.com/guide/topics/ui/themes.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics)!

For more information about styles resources, check out the documentation [**here**](http://developer.android.com/guide/topics/resources/style-resource.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics) and [**here**](http://developer.android.com/guide/topics/ui/themes.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics). Note that styles.xml is in a resource folder. This means you can override the style and provide different attributes on differently sized devices. You just need to create a new styles.xml file, within the appropriate values folder. For example, if you wanted a HeaderTextStyle to show bigger text if the user is on a tablet with the smaller side’s screen width of 600 dp or greater, you could save a new version of the HeaderTextStyle in res/values-sw600dp/styles.xml folder. Check out this[**documentation**](http://developer.android.com/training/multiscreen/screensizes.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics#TaskUseSWQuali) for more information. Just Java app adhering to Material Design spec: The height of 48dp for a header comes from the [**Material Design spec for Single-line list**](https://www.google.com/design/spec/components/lists.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics#lists-specs) (see title height). We chose 15sp from the recommended text size for body text[**here**](http://www.google.com/design/spec/style/typography.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics#typography-styles). Our checkboxes are already implemented according to the Material Design guidelines. See "Icon with text" guidelines on this [**page**](https://www.google.com/design/spec/components/lists.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics#lists-specs). The checkbox is treated as the icon in this case. The checkbox should be located 16dp from the left edge of the screen. The text for the checkbox should be located 72dp from the left edge of the screen. These are the two major keylines (at 16dp and 72dp from the left edge) recommended in the [**spec**](http://www.google.com/design/spec/layout/metrics-keylines.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics#metrics-keylines-keylines-spacing). Another helpful tool to verify that your app is aligned to the [**8dp baseline grid**](http://www.google.com/design/spec/layout/metrics-keylines.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics), is to use the [**Keyline Pushing app**](https://play.google.com/store/apps/details?id=com.faizmalkani.keylines&hl=en), available on Google Play. If you turn on the grid, and then go back into your app, you can see that for the most part, our content lines up with the 16dp and 72dp keylines. If it doesn’t line up exactly, it’s because there is a slight amount of padding built into the default assets provided by Android (like the checkbox).

Have questions? Head to the [**forums**](https://discussions.udacity.com/c/standalone-courses/android-beginners) for discussion wi

22\_**Quiz: Themes**

Click [**here**](https://developer.android.com/training/material/theme.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics) to learn more about the Material system themes.

Here’s an [**example code snippet**](https://developer.android.com/training/material/compatibility.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics) for defining a theme that is compatible with older versions of Android.

To learn more about AppCompat, check out Ian Lake’s DevByte on the Google Developers YouTube Channel:[**https://www.youtube.com/watch?v=5Be2mJzP-Uw**](https://www.youtube.com/watch?v=5Be2mJzP-Uw)

What is the [**AndroidManifest.xml file**](http://developer.android.com/guide/topics/manifest/manifest-intro.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics)?

Android For Beginners Course Survey

Thank you for your feedback and participation in this course! The survey code is: I<3ANDROID

## 24\_So Many Things to Explore in Android

As we near the end of Android for Beginners, we sincerely hope you've enjoyed this wild, crazy ride in app development with us. You're probably already noticing that we've just covering the tip of the iceberg. So where to next?

### Where to go next

[**Android Basics: Multi Screen Apps**](https://classroom.udacity.com/courses/ud839) is currently our next Android Course (and the next course in the [**Android Basics Nanodegree by Google**](https://www.udacity.com/course/nd803)). This course covers how to build more complex android apps and also teaches more object-oriented programming concepts.

Another resource is the [**Java Language tutorial**](https://docs.oracle.com/javase/tutorial/) which we've referenced throughout this course. Also, share you tips for good introductory and intermediate Java material [**in the forums**](https://discussions.udacity.com/t/what-do-you-want-to-learn-more-about/24920).

You can find the final code for Just Java on [**this Github web-page**](https://github.com/udacity/Just-Java). [**Github**](https://github.com/) is a website that facilitates sharing code online. Code on Github is organized via the [**Git**](https://git-scm.com/) version control system. To learn more about Git and Github, consider taking our [**version control course**](https://www.udacity.com/course/how-to-use-git-and-github--ud775), which is another good introductory offering.

### Advanced Resources and The Possibilities

If you're wondering what the possibilities are for Android Apps, there is a lot of material you can look into. Much of it will require knowledge from Developing Android Apps, so these are considered pretty advanced resources.

What follow are some ideas for deeper dives:

**Build for tablets** - We've be focusing on designs for phones, but with resource files, you can actually create custom layouts for your larger devices, such as tablets. Here's some[**documentation**](http://developer.android.com/guide/practices/tablets-and-handsets.html) to get you started.

**Create lists** - You'll often find android apps that have long lists of items a user could click on - think of messages in an inbox, or stories in a news feed. Creating lists like this is greatly optimized when someone used a view like [**RecyclerView**](https://developer.android.com/training/material/lists-cards.html#RecyclerView).

**Create cards** - Cards, not to be confused with your awesome Birthday Card, are a special kind of Material Design inspired view that has rounded corners and a slight shadow. If you've ever used Google Now, you've seen a card. Check out this [**documentation**](https://developer.android.com/training/material/lists-cards.html#CardView) to see how to add card elements to your app.

**Load up things from the internet** - Want to load text or images from the internet? Check out the[**volley library**](https://developer.android.com/training/volley/index.html), for general purpose loading of web data. [**Glide**](https://github.com/bumptech/glide) is a great choice for loading images from the web. You can check out [**this lesson**](https://classroom.udacity.com/courses/ud855/lessons/4025488663/concepts/43549085380923) from the Libraries lesson of Advanced Android Development to see how to incorporate libraries like these.

**Store data on the device** - Does your app need any information saved between launches? Maybe you want to keep a history of all coffees bought with JustJava. To do this, you would need to save some data associated with the app. Check out the [**Android documentation**](http://developer.android.com/guide/topics/data/data-storage.html) on data storage for your options.

**Post notifications** - [**Notifications**](http://developer.android.com/guide/topics/ui/notifiers/notifications.html) are messages that the user sees outside of your app. They appear in the status bar. You can then pull the status bar down to see more details about the notification. When you get a text message, for example, many times your phone will beep and show you a notification so that you can easily view the text. To learn more about designing notifications, check out this [**Android guide**](http://developer.android.com/design/patterns/notifications.html).

**Do background operations** - Perhaps you want to create an app that does something, even when it's not visibly on the screen. For example, a music player application, that plays music as you so other things on your phone. Or a messaging application that pulls down the latest chats sent to you. For this, you'll need to learn more about [**creating services.**](https://developer.android.com/training/run-background-service/create-service.html)

**Use Google Play services** - Using Google Play Services is a library of Google code that gives you access to popular functionality, such as phone location, authentication (information and ability to “log in") and even fitness data. Check out some of the recent updates [**here**](https://www.youtube.com/watch?v=M3Udfu6qidk).

**Use location** - Access to location is one of the many capabilities offered by Google Play services. By figuring out where a user is physically located, your app can give them more data about their surrounding and customize itself to their specific needs. Google Maps, for example, can find nearby restaurants in this manner. Check out our class on [**Location and Context**](https://www.udacity.com/course/google-play-services-location-context--ud876-1), which is part of a larger series of mini courses on all of the Google Play Services.

**Add analytics** - Ever wondered who is using your app? Or how they are using it? By collecting this data you can make informed decisions about what features to add, update or fix. Analytics is also included in the Google Play Services libraries. Check out our course on [**Analytics**](https://www.udacity.com/course/google-play-services-analytics--ud876-2).

**Build for other form factors** - It doesn't stop at tablets; the android operating system (and therefore your apps) run on watches, TVs, even in cars. If it makes sense, you can have different versions of your app for these different “form factors". The Maps App, for example, can give you directions on your phone, or your watch, or while you're driving around. The base functionality is the same, but it's been re-designed and tweaked for each interface. Check out our [**Ubiquitious Design**](https://www.udacity.com/course/android-ubiquitous-computing--ud875) course.

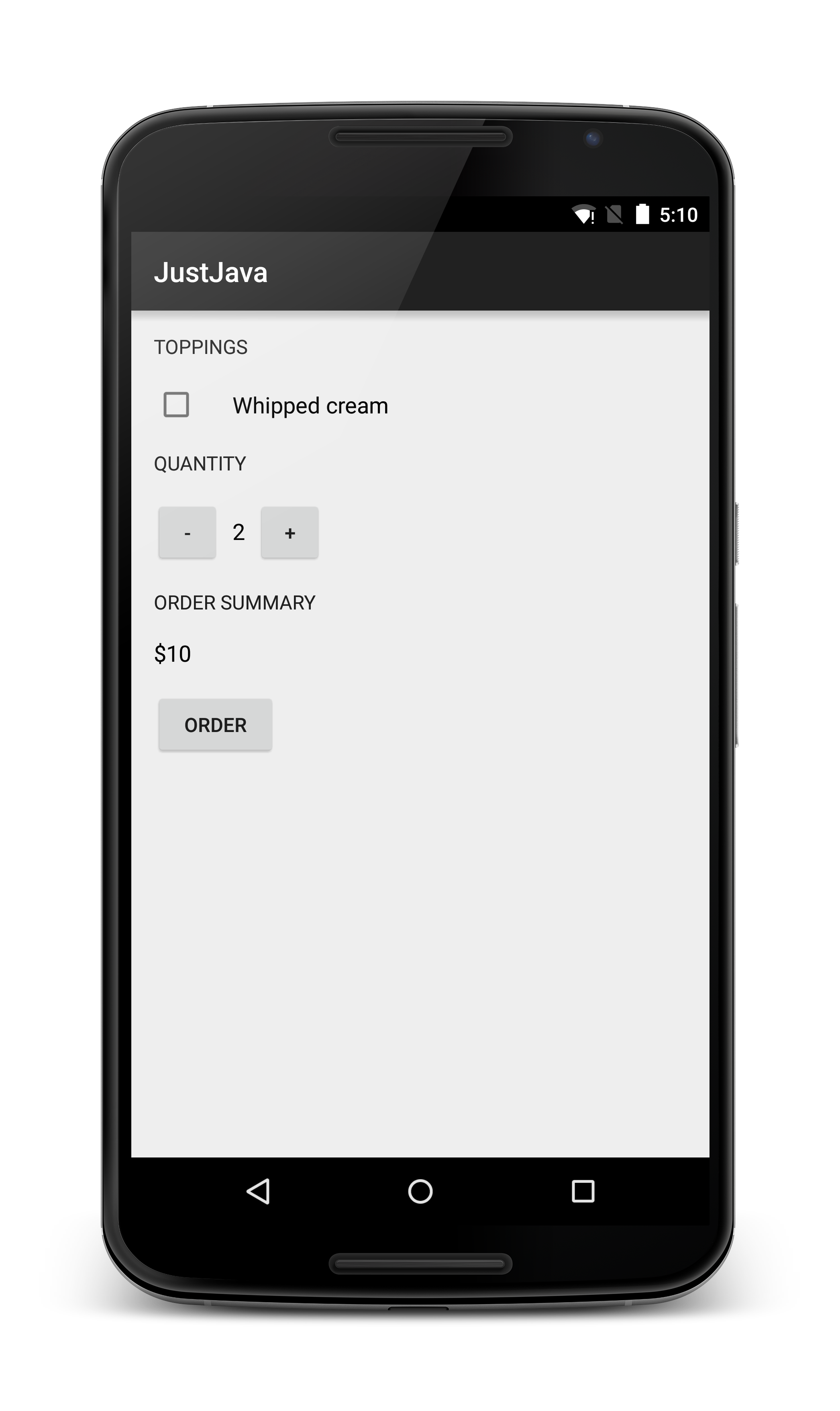
**Play sound** - Add sounds to your apps, whether is be short clips for a game or playback capabilities for a podcast app. Check out Google's guide to [**Media Playback**](http://developer.android.com/guide/topics/media/mediaplayer.html).

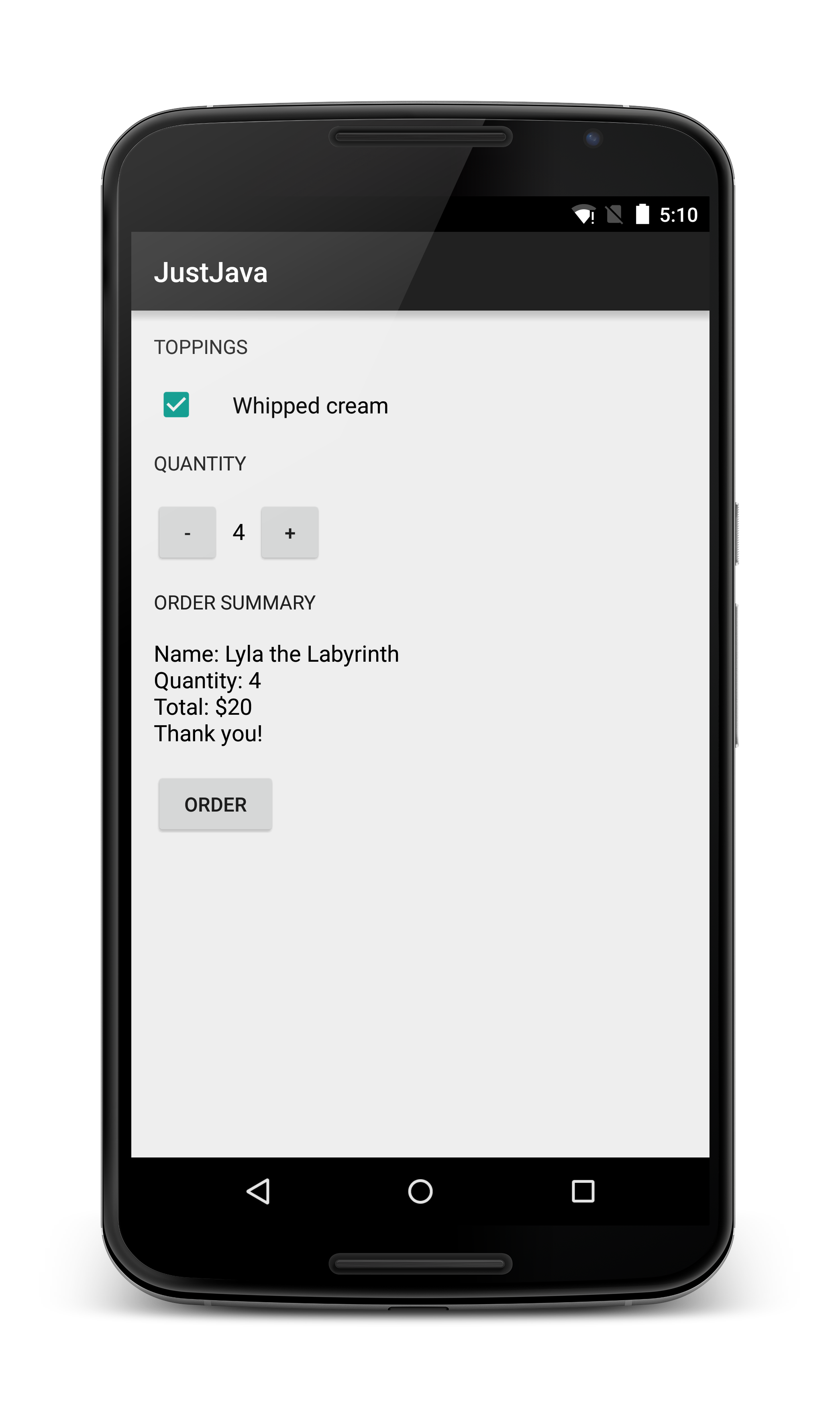
**Animations** - Subtle animations that give your views a real sense of space and physical existence are an integral part of material design. Check out the [**documentation**](http://developer.android.com/training/animation/index.html) on animations to add things like cross fading, zooming and flipping animations to your layouts.

**Create server backend for your app** - A stand alone app is great, but what makes an app really powerful is if it is connected to your own web server. Web servers can help organize and process data from across the world and send it to your user's phone. Any Google App more complicated than the alarm is accessing some sort of google server. Servers store everything from your Calendar events, to your friend's G+ photos. Learn more about building a scalable web server in[**Java**](https://www.udacity.com/course/developing-scalable-apps-in-java--ud859) or [**Python**](https://www.udacity.com/course/developing-scalable-apps-in-python--ud858).

**Fitness** - Ever wanted to make an app that will help users take control of their health and fitness? Google Fit is Google's open platform with user fitness data. Check out the [**documentation**](https://developers.google.com/fit/).

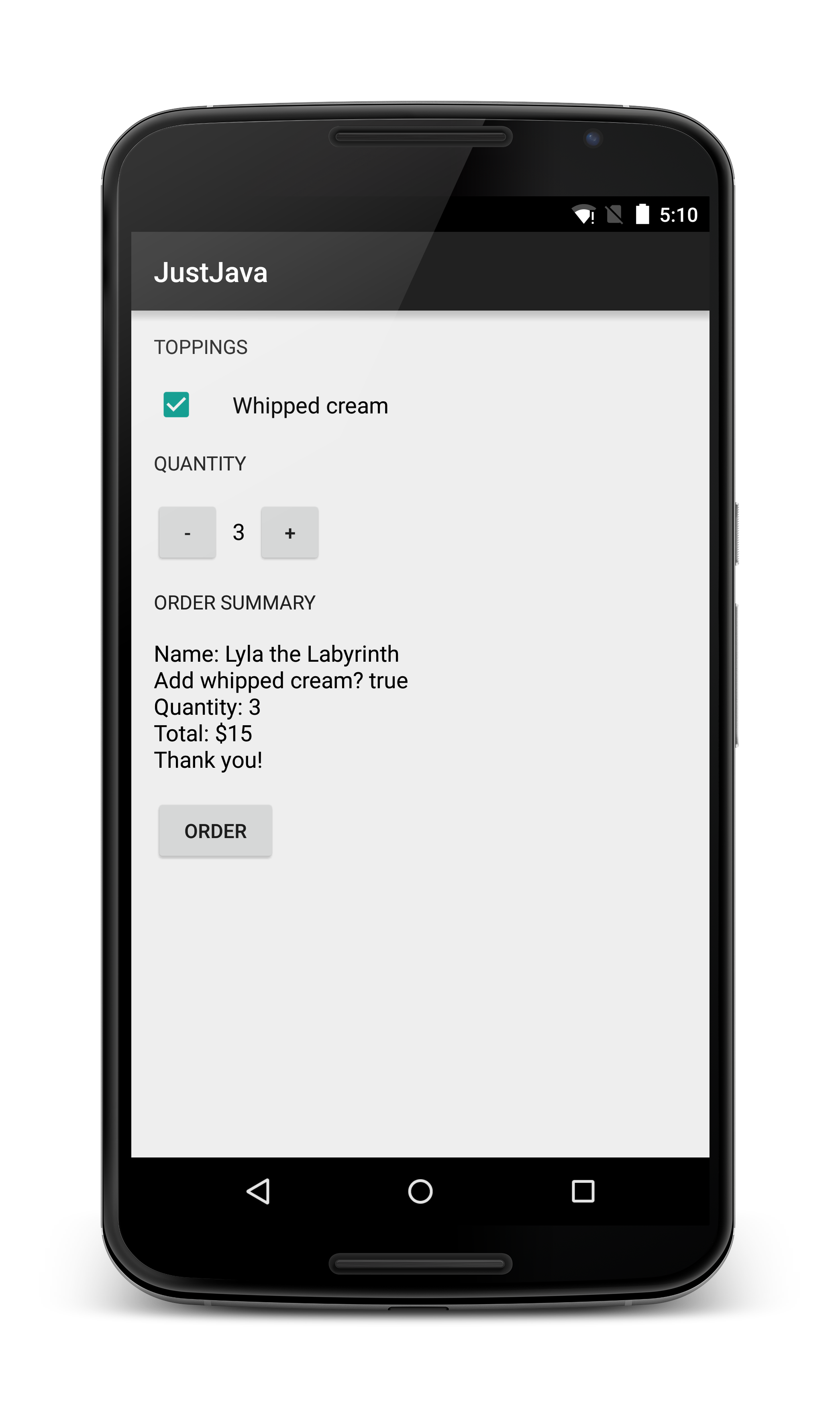
**Test out the latest M Developer Preview** - To get the bleeding edge latest version of the Android platform, you can try out the [**Android M Developer Preview**](https://developer.android.com/preview/index.html). We make an early version of the platform available to developers so they can prepare and test their app before the M release launches to the public. Check out this [**video**](https://www.youtube.com/watch?v=dcmNjbCNvA8).

**Screenshot of app (on app launch)** 

**Screenshot of app (after hitting the “Order” button)** 

[**Link to Common Android Views cheat sheet**](https://s3-us-west-1.amazonaws.com/udacity-content/PDFs/Common+Android+Views+Cheat+Sheet+(1).pdf)

3\_**Quiz: Update Order Summary**

**End Goal** 

Here's some more detailed info for each step:

* When the button is clicked, find the CheckBox view, get checked state from the CheckBox, and store the checked state value in a new boolean variable. Feel free to add a log message to verify that step 1 is completed correctly.
* Pass the checked state boolean into the createOrderSummary() method, so it takes in 2 input parameters. The new input parameter is a boolean called hasWhippedCream. Remember to modify the method signature of the method. Here's a link to [**the video**](https://classroom.udacity.com/courses/ud837/lessons/4619208555/concepts/45934490260923) discussion adding and removing parameters.
* Modify the createOrderSummary() method so it displays this text on screen using the boolean input parameter.
* Remove log messages that were added for debugging purposes.

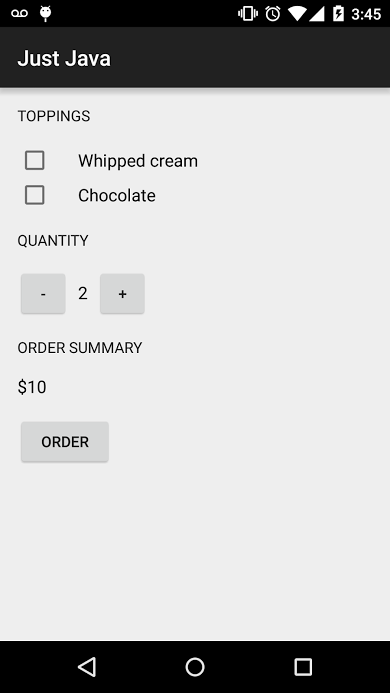
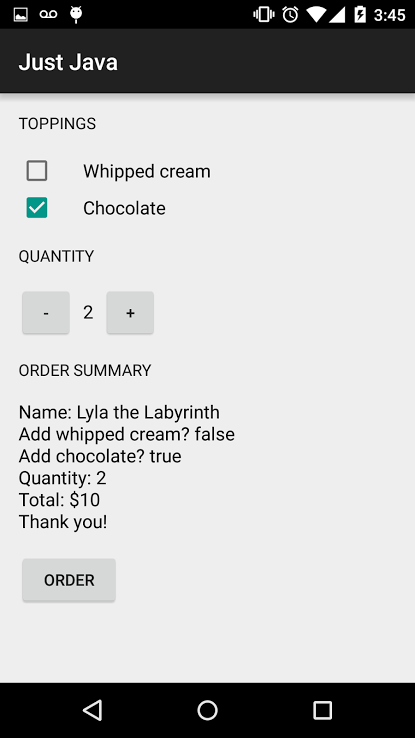
## 5\_Add the Chocolate Topping Checkbox

Things are getting tougher! We’re offering less hand-holding, and letting you come up with the code yourself. In this task, you will implement a whole feature on your own.

**Your Goal**

Your goal is to add a chocolate topping checkbox in the Just Java app.

This will be similar to the whipped cream topping checkbox. This is what the app should look like when you’re finished.

**How to Start**

In the real world, you’ll get requirements about what the app should do, and you will have to come up with all the steps to make the app reach that goal. To start, plan out the steps for how you will need to modify the code.

If you need help, go back and watch how we added the [**whipped cream checkbox**](https://classroom.udacity.com/courses/ud837/lessons/4584545214/concepts/TODO) to the app.

Every time you make a change to the code, stop and make sure your app still runs on your device and does what you expect. If you run your app after every change and you ever find an unexpected problem, it’s obvious what change caused the bug. If you've made twenty changes since you last ran your app, you'll have a very hard time finding the source of any problems.

SKIP TO QUIZ

5\_solution

## Add the Chocolate Topping Checkbox Solution

### Step 1 : Change the XML

First, let’s update the XML. We want to add a checkbox in the LinearLayout. That can be done with the code [**here**](https://gist.github.com/udacityandroid/273e1273286e5169c87d).

Specifically, we add the following lines:

<CheckBox

android:id="@+id/chocolate\_checkbox"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:paddingLeft="24dp"

android:text="Chocolate"

android:textSize="16sp" />

### Step 2 : The Plan for the Java Changes

Given the new checkbox, what should happen when the submit button is clicked? Let’s think about the final pseudocode for what will happen when the submit order button is clicked.

**When the submit order button is clicked**

1. Set hasWhippedCream to the value of the checked state in the whipped cream CheckBox.
2. **Set hasChocolate to the value of the checked state in the chocolate CheckBox**.
3. Calculate the price and store this value in the price variable.
4. Create the order summary and store this String in the message variable.
5. Display message on the screen.

**Creating the order summary**

1. Set priceMessage to the name.
2. Append to priceMessage "Add whipped cream?" and the value of whether the coffee has whipped cream.
3. **Append to priceMessage "Add chocolate?" and the value of whether the coffee has chocolate.**
4. Append to priceMessage "Quantity:" and the quantity.
5. Append to priceMessage "Total: $" and the total price.
6. Append to priceMessage "Thank you!"
7. Return the priceMessage

The steps in bold are the ones that need to be implemented or fully implemented given that you’re adding this new checkbox.

### Step 3 : Change the Java

**Set hasChocolate to the value of the chocolate CheckBox**

In submitOrder method, we need to get the checked state from the chocolate CheckBox. Here are the steps for doing that:

1. You don’t have a reference to that object, so you’ll need to get the CheckBox object using findViewById. You’ll need to pass the chocolate checkbox id.

findViewById(R.id.chocolate\_checkbox);

1. You can then store the returned value in a Checkbox object.

CheckBox chocolateCheckBox = findViewById(R.id.chocolate\_checkbox);

1. Because findViewById returns a generic View, you’ll need to cast it to a CheckBox.

CheckBox chocolateCheckBox = (CheckBox) findViewById(R.id.chocolate\_checkbox);

1. Now you’ll need to call a method on the object. In the [**documentation**](http://developer.android.com/reference/android/widget/CompoundButton.html#isChecked()) you’ll see there is a method calledisChecked.
2. Save what is returned from isChecked in a boolean variable.

boolean hasChocolate = chocolateCheckBox.isChecked();

The final code for getting the value of the checkbox and storing it in a variable is:

// Figure out if the user wants chocolate topping

CheckBox chocolateCheckBox = (CheckBox) findViewById(R.id.chocolate\_checkbox);

boolean hasChocolate = chocolateCheckBox.isChecked();

**Creating the order summary**

The only issue here is that you need access to the boolean you just retrieved in the createOrderSummary method. We’ll need to pass the boolean to createOrderSummary method as an input parameter.

1. Here is the code when you **call** the createOrderSummary method that passes in the hasChocolate boolean:

createOrderSummary(price, hasWhippedCream, hasChocolate);

1. This will produce an error in Android Studio because the method call does not match the method definition (the number of parameters is different). Therefore, you also need to change where the method is **defined** so that it has this new parameter.

private String createOrderSummary(int price, boolean addWhippedCream, boolean addChocolate)

1. Update the javadoc comment as well to document this new input parameter addChocolate.
2. /\*\*
3. \* Create summary of the order.
4. \*
5. \* @param addWhippedCream is whether or not the user wants whipped cream topping
6. \* @param addChocolate is whether or not the user wants whipped cream topping
7. \* @param price of the order
8. \* @return text summary
9. \*/

**Append to priceMessage "Add chocolate?" and the value of whether the coffee has chocolate.**

Okay, now you need to change the message that is displayed to show whether the customer ordered chocolate or not. The string to do that is below:

"Add chocolate?" + addChocolate

The final code for create order summary that appends this string is:

private String createOrderSummary(int price, boolean addWhippedCream, boolean addChocolate) {

String priceMessage = "Name: Lyla the Labyrinth";

priceMessage += "\nAdd whipped cream? " + addWhippedCream;

priceMessage += "\nAdd chocolate? " + addChocolate;

priceMessage += "\nQuantity: " + quantity;

priceMessage += "\nTotal: $" + price;

priceMessage += "\nThank you!";

return priceMessage;

}

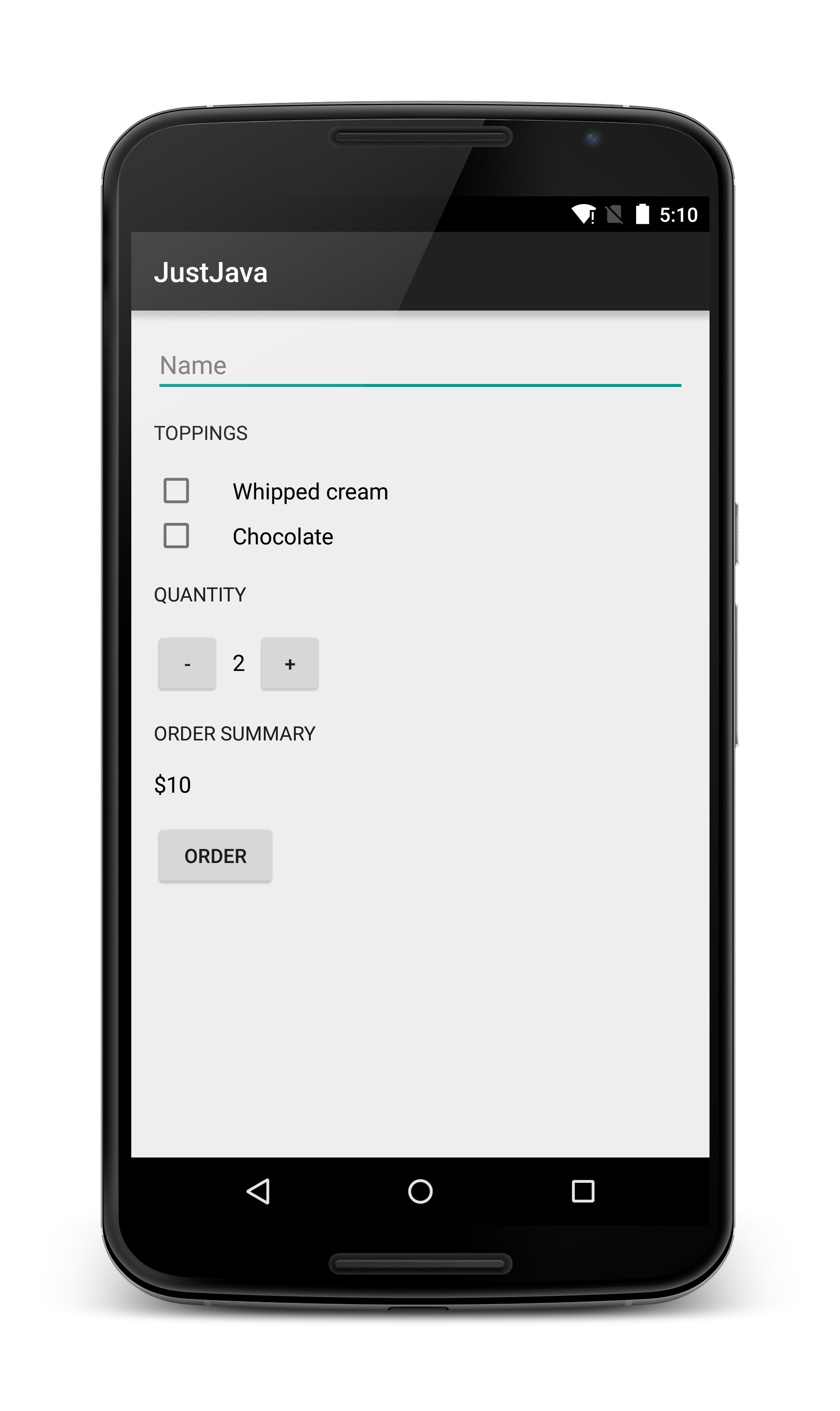
### The Final Code

The final java code can be found [**here**](https://gist.github.com/udacityandroid/609dbb5370ba0665955a).

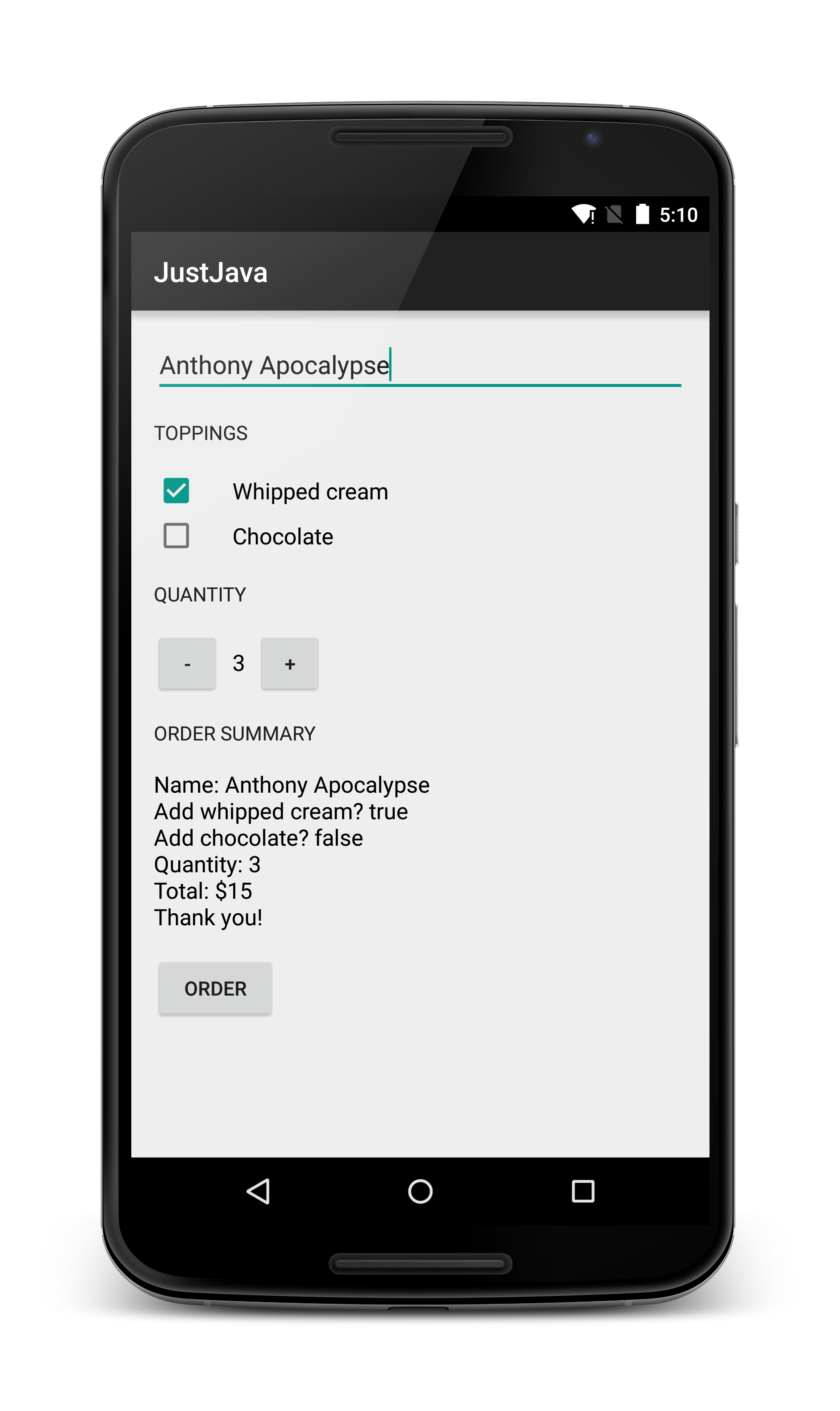
8\_**Quiz: What's Your Name?**

[**Common Android Views Cheat Sheet**](https://drive.google.com/a/udacity.com/file/d/0B5XIkMkayHgRMVljUVIyZzNmQUU/view)

**This is what the app should look like on launch after updating your code**



**This is what the app should look like after you click the ORDER button**



## 14\_Equality and Relational Operators

We’ve used some symbols like < => and == in our examples. These are known as **Equality and Relational Operators**. These operators work a lot like the math or arithmetic operators we’re used to (+ - / \*). Arithmetic operators take two numbers and evaluate to another number. For example 5 + 3 evaluates to 8.

Equality and Relational Operators take two values (these are usually but not always numbers) and compare the values. They then evaluate to a boolean. They look like symbols that you might have used in math class to talk about comparing numbers.

**The Equality and Relational Operators in Java**

* == - Equal to (note that it is not a single = sign because this is the **assignment** operator)
* != - Not equal to
* > - Greater than
* >= - Greater than or equal to
* < - Less than
* <= - Less than or equal to

**Example**

Here's an example of how expressions using relational operators work:

5 < 10 is an expression meaning 5 is less than 10. This evaluates to the value **true**, because 5 is less than 10.

5 == 5 evaluates to true.

5 != 5 evaluates to false. This is because it is false that five is not equal to five.

Much more could be said about relational and equality operators. There is a lot of good information out on the web about these concepts. A good place to start is the [**Oracle Java tutorial about operators**](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html) and [**Java basic operators**](http://www.tutorialspoint.com/java/java_basic_operators.htm).

16\_**Quiz: Negative Cups Of Coffee**

Hint: You can put a ‘return’ keyword inside of an if statement, and that that will end the method early. See the highest rated answer in this [**StackOverflow post**](http://stackoverflow.com/questions/7937029/how-to-break-out-or-exit-a-method-in-java).

Hint: Check [**this guide**](http://developer.android.com/guide/topics/ui/notifiers/toasts.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics) on how to pop up toast messages in an app.

[**Solution Code**](https://gist.github.com/udacityandroid/2d4afee7c69adc6df544)

19\_**Quiz: Localization (OPTIONAL)**

[**Code snippet**](https://gist.github.com/udacityandroid/759b8b4c9ed9e6806e90) of Spanish translations for Just Java app

See [**this article**](https://developer.android.com/distribute/tools/localization-checklist.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics#manage-strings) for an in-depth explanation of how to manage strings.

You can also refer to [**this video**](https://classroom.udacity.com/courses/ud853/lessons/1623168625/concepts/16677585930923) in the “Developing Android Apps” course about formatting strings.

Alice Yang, who we sat down to chat with in Lesson 2, created a [**video**](https://www.youtube.com/watch?v=_b0iU2LzlUo) on how to localize a Google Glass app. Google Glass is based on the Android platform, so these tips on localization apply to Android apps as well!

[**Localization checklist**](http://developer.android.com/distribute/tools/localization-checklist.html?utm_source=udacity&utm_medium=course&utm_campaign=android_basics) before launching your app