

Lesson 3.3: Contacts in an App

ACTIVITY 3.3.1

Trip Cost and Rating

INTRODUCTION

In this activity, you will add new features in TripTracker:

- The capability to store the cost of a trip
- The capability to store a rating for the trip

Once you have this data stored for each trip, you will be able to:

- Search for highly rated trips
- Apply a discount to trips if you happen to be a frequent traveler

Materials

- Computer with BlueJ and Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student
- A Google account and a project on the Google API Console



Procedure

Part I: Prepare the Project

- 1 Open your TripTracker app in Android Studio.

NOTE

If you were unable to complete the last TripTracker activity from Lesson 3.2, *Activity 3.2.2 Using Google Play Services*, import *3.2.2TripTracker_Solution* as directed by your teacher. Recall that if you import the solution, you must update keys values in *strings.xml*:

- Change `be_app_id` and `be_android_api_key` to your Backendless App ID and Android API key values, respectively. You can retrieve these from your  **Backendless Console (Manage icon)**.
- Change the value for `google_app_id` to your Google Play service API key. You can retrieve it from the  **Google API Console (Credentials)**.

- 2 Open *fragment_trip.xml* and add the new `TextView` and `EditText` views for cost and rating in a relative layout. Insert this after the trip's description and before the, public check box.

```
1 <!--cost and rating -->
  <RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
      android:id="@+id/trip_details_cost_label"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/trip_details_cost"
      android:paddingLeft="@dimen/spacer"
      style="?android:listSeparatorTextViewStyle" />
    <EditText android:id="@+id/enter_trip_cost"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:inputType="numberDecimal"
      android:paddingLeft="@dimen/spacer"
      android:layout_below="@id/trip_details_cost_label"
      android:hint="@string/trip_cost_hint" />
    <TextView
      android:id="@+id/trip_details_rating_label"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/trip_details_rating_label"
      android:layout_below="@id/trip_details_rating_label"
      android:layout_alignParentEnd="true"
      android:paddingLeft="@dimen/spacer"
      style="?android:listSeparatorTextViewStyle" />
    <EditText android:id="@+id/enter_trip_rating"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```

        android:layout_below="@id/trip_details_rating_label"
        android:inputType="number"
        android:layout_alignParentEnd="true"
        android:gravity="end"
        android:paddingLeft="@dimen/spacer"
        android:hint="@string/trip_rating_hint" />
    </RelativeLayout>

```

- 3 To make room for the new widgets, reduce the height of all of your buttons by adding `android:minHeight="1dp"` to their definitions.
- 4 Add the necessary string values to `strings.xml`.

```

1 <!-- 3.3.1 cost and rating -->
  <string name="trip_details_cost">Cost</string>
  <string name="trip_cost_hint">Estimated</string>
  <string name="action_calc_ave">Ave Cost</string>
  <string name="trip_details_rating_label">Rating</string>
  <string name="trip_rating_hint">(1-5)</string>
  <string name="action_rate">Find Highest Rated Trips</string>

```

- 5 Add menu items to the My Trips options menu (`menu_trips.xml`) to average costs and find the highest rating.

```

1 <!--3.3.1 cost and rating -->
<item
    android:id="@+id/action_calc_ave"
    app:showAsAction="ifRoom|withText"
    android:title="@string/action_calc_ave"
    android:icon="@drawable/ic_menu_dollar">
</item>
<item
    android:id="@+id/action_rate"
    app:showAsAction="ifRoom|withText"
    android:title="@string/action_rate"
    android:icon="@drawable/ic_menu_star">
</item>

```

- 6 You should notice that you need some drawable icons for these menu items. Get a copy of `3.3.1TripTracker_StarterCode` from your teacher and copy or extract the files to a temporary location.

- 7 Copy `ic_menu_dollar.png` and `ic_menu_star.png` to your project's `res/drawable` folder.
- 8 Compile your app. Your UI should now have the following:
 - a. Two new items on the My Trips option menu. You can see the icons if you have room (`ifRoom|withText`) when your device or emulator is horizontal.
 - b. The cost and rating views on Trip Details.
- 9 Test whether the UI elements are enabled and responsive. (They will *not* actually do anything yet, that is the next part of this activity.)

Why can you only enter numbers for rating, and only numbers and a decimal point for cost?

Part II: Javadocs

- 10 Open `Trip.java` and create variables and their getters and setters for the trip's cost (a double) and its rating (an int). This activity refers to these variables as `cost` and `rating`.

Until now, you have not commented your `Trip` class. The standard convention for commenting Java classes and methods is called **Javadocs**. Examples of Javadocs for the `getOwnerId` and `setOwnerId` methods are shown below.

Javadoc

A document generator built into the Java compiler that automatically generates documentation in an online format.

```
/**
 * Returns the ownerId of the user who owns the Trip. This is derived
 * from the user's objectId in the Backendless User table.
 *
 * pre-conditions: ownerId is not null
 *
 * @return the trip's owner
 */
public String getOwnerId() {
    return ownerId;
}

/**
 * Sets the owner ID of this trip, which is a userVC from the
 * Backendless
 * User table. The user's objectId is used as the trip's ownerId.
 *
 * @post-condition: the user who owns the trip has a valid objectId
 *
 * @param ownerId The objectId of the user who owns the trip
 */
public void setOwnerId(String ownerId) {
    this.ownerId = ownerId;
}
```

- 11 Javadocs begin with `/**`, and when you type this in an Android Studio Java file, the `@param` and `@return` keywords are added when appropriate.

What do you think the `@param` and `@return` tags designate?

- 12 Write Javadocs for the methods in your `Trip` class, specifying at least the `@param` and `@return` tags where appropriate.

What other methods do you predict you will need for *Trip.java*?

As shown in the examples above, Javadocs also use a pre-condition and post-condition nomenclature that informs the reader of any pre-existing conditions that must be met and any consequences or post-conditions that occur in the method. They are usually written for more complex and involved methods such as your `TripFragment`'s `updateTrip` method.

- 13 For `updateTrip` in `TripFragment`, add a pre- and post-condition in the Javadocs.
- For the pre-condition, observe `mTrip` and determine what might cause the method or app to terminate or crash. This would be a pre-condition of the method that must be met.
 - For the post-condition, ask yourself what is happening behind the scenes that someone may need to know. Hint: Let the reader know what is getting updated and how it is “updated”.

Part III: Autobox and Auto-unbox

`TripFragment` needs new functionality for trip cost and trip rating. This requires using the new UI elements and intent data.

- 14 In *IntentData.java*, add intent constants for the trip's cost and rating.
- 15 Open *TripFragment.java*, and in `onCreate`, get the cost and rating data from the intent and save it to the trip. (You will *put* this data on the intent in a later step.)
- Create local variables `double cost` and `int rating`.
 - Use the `getDoubleExtra` and `getIntExtra` methods.
 - Assume a default of 0 (zero) for both values.
 - Set these values for the current trip, `mTrip`.

- 16 Review the slideshow below of the *3.3.1 Trip Cost and Rating* presentation.

The `intent` methods you just wrote return `Double` and `Integer` objects, yet you store these in `double` and `int` variables, respectively. Explain how **auto-unboxing** is used to do this.

auto-unbox

A process that automatically converts numeric and Boolean objects to their primitive versions.

- 17 Add the functionality to `onCreateView` to get the cost and rating from the trip (`mTrip`) and display it on the Trip Details screen.

Since the UI elements use `setText` with character data (and not integer or double values), you will need to convert the primitive data types to an object, such as `Double` or `Integer`, and use their `toString` methods.

- Create variables and get reference IDs for the cost and rating `EditText` elements.
- Create object versions `Double` and `Integer` of your cost and rating values from the trip. Explain how **autoboxing** was used.
- Use `toString` method on the `Double` and `Integer` objects to set the values for the cost and rating `EditText` elements.
- Test your app to see that the values for cost and rating are “0.0” and “0”, respectively.
- Cost doesn’t look like a monetary value, so to display the cost in a user-friendly money format, use a new `DecimalFormat` object.

auto-unbox

A process that automatically converts numeric and Boolean objects to their primitive versions.

```
1: DecimalFormat df = new DecimalFormat("#.00");
2: mCostField.setText(df.format(cost));
```

Cost should appear as .00.

Currently, no one has entered data for the cost or rating of a trip, and when the values are 0, the user loses the hint that appeared in the `EditText` field.

Autoboxing

- **Autoboxing** automatically converts primitives to objects (this is called *promotion*)
- Programmers often need object version of data to gain useful methods
- Example:

```
double val = 3.14;  
Double dVal = val; // autobox occurs here
```
- With the object, you gain useful methods such as:

```
dVal.toString();  
dVal.equals(3.14159);
```
- `int` and `Integer` work the same way

Computer Science A

© 2016 Project Lead The Way, Inc.

When programming user interfaces, often you need to convert numerical data to string representations (and vice versa). Java provides `Double` and `Integer` classes with useful methods such as `toString()`.

In the `Double` class, programmers gain methods that can be used to manage the “double” data. In `TripTracker`, you will be most interested in converting a double data type to a string via the `toString()` method of the `Double` class.

Because converting between double and `Double` is used so often, Java provides an automatic conversion process called “autoboxing”. When a double primitive type is automatically promoted to a `Double`, it is autoboxed, and you can use `Double` methods, such as `toString()`, `equals()`, and a variety of others.

An `int` can be autoboxed to an `Integer` in the same way.

Auto-unboxing

- Works in reverse
- Provides automatic demotion from object to primitive

```
Double dVal = new Double(3.14);  
double d = dVal; // auto-unbox occurs here
```
- A useful algorithm for managing UI data

```
// assume field contains numeric data  
String s = (String)field.getText();  
Double dVal = new Double(s);  
double d = dVal; // auto-unbox occurs here
```

Computer Science A

© 2016 Project Lead The Way, Inc.

You can use auto-unboxing to retrieve information from a UI. In most UIs, the values in text boxes are represented as `Strings` in code, and need to be converted to numeric values such as integers and doubles. The algorithm shown here retrieves a string value from a field, creates a `Double` variable based on the value in `s`, and then auto-unboxes the `Double` to a double.

- 18 Modify your code so this does not happen and confirm that the hints will show.
- 19 Test with a publicly shared trip that the user does *not* own and determine whether the cost and rating fields are enabled. If necessary, modify your code to disable these elements when the trip's owner is not the current user.

The last step is to save a trip's cost and rating information.

- 20 In `updateTrip`, get the data from the view for cost and rating.
 - a. Create local variables `double cost` and `int rating` to match the variables types of `cost` and `rating` in `mTrip`.
 - b. As you had to do with setting the data, you will need to convert the values from `Strings` to a `Double` or `Integer` data type and use the auto-unbox feature of Java. For the cost, this is done with the following:

```
cost = new Double(mCostField.getText().toString());
```
 - c. Get the rating in a similar way.
- 21 Test your app with some data entered for cost and rating and save your trip.
- 22 Check to be sure the values were saved in Backendless.

When you load your trip again, why do you think cost and rating are not showing on the Trip Details screen?
- 23 Test your app with empty data for cost and/or rating. Your app should crash!

What is the exception that was thrown?

The error occurred because your code attempted to convert the empty string to a numeric value (either a `Double` or an `Integer`).
- 24 Use a `try/catch` statement around the “dangerous” code. When a user attempts to save an empty value, log a message and assign cost or rating a “0” value.
- 25 Finally, validate input for the rating for values 0 – 5. If invalid data is entered, display a `Toast` message and return. This will allow the user to correct their input. Be sure to place this code *before* the code that change the progress bar.
- 26 Test your app:
 - a. Attempt a save with an invalid value for rating.
 - b. Save the trip with valid data. (You will not see the data when you select this trip from the My Trips list yet.)

Part IV: Backendless

- 27 Log in to the Backendless Console and confirm that the values for cost and rating were saved.

Backendless automatically created the columns for cost and rating when you saved a trip, but notice, the data types are not quite right.

- 28 Change the data type for rating to accurately reflect the real data type of rating, using the **SCHEMA** page. To save changes, click the pencil **Save** icon to the far right at the end of the row.

- 29 Your trip is saved with cost and rating data in Backendless, but is not showing when you select the saved trip from your My Trips screen. Correct this omission.

Is autoboxing or auto-unboxing used here? Why or why not? You may have to refer to the Android Developer's reference to answer this question.

Part V: Averaging and Searching

Now that your trip has a cost and a rating, you can write the functionality for the menu items that were added to the TripList menu: Average the cost of all trips and find the highest rated trip.

- 30 If you do not have three or four trips with a variety of ratings and costs, create them now. Then, modify `TripListFragment` in the following ways:
- Using an enhanced for loop, add the functionality to find the highest rated trips (5 should be considered the highest). Inform the user in a toast (`LENGTH_LONG`) and include the rating. If there is more than one, be sure to concatenate the name of all trips into the toast.
 - Review the slide below of 3.3.1 Trip Cost and Rating presentation. Add the functionality to average the cost of all trips, including a discount where every third trip you take will receive a 15% discount.
 - Do not hard code the values “3” and “15%” into your algorithm; rather, create constants for these values.
 - Consider that you do not need to look at all of the trips in your array list, only every third. Use the best loop construct to do this.
 - The algorithm should use the **modulus operator** and not an additional counting variable.
 - Use logging messages to confirm that you have discounted the correct trip, then manually calculate the expected result to ensure your algorithm is working properly.

modulus operator

Denoted with a percent sign, %, it calculates the remainder of integer division.

- c. Describe in your own words how you found the highest rated trips and averaged the costs.

If one of your trips does not have a cost, it does not make sense to get the average cost of all trips since the result would be greatly skewed.

- 31 Modify your Average Cost algorithm to prevent this incorrect average from showing and decide what to display when one or more trips does not have a cost.

Modulus

- %
- A useful operator whose result is the *remainder* of a division operation

```
int r;  
r = 9 % 5; // r is 4  
r = 5 % 2; // r is ?  
r = 26 % 3; // r is ?  
r = 16 % 4; // r is ?
```

Computer Science A

© 2016 Project Lead The Way, Inc.

The modulus operator, %, returns the remainder value of integer division.


Predict the value of r in the examples.




Part VI: Regression Testing

You have added a lot of new functionality to TripTracker. It is a good time to check previous functionality still works. This is called *regression testing* and is a good idea when many changes have been made to your code.

- 32 Test all of the functionality, including signing up a new user, attempting false logins, and creating, changing, and deleting trips with locations, cost, and rating.
- 33 (Optional) When you navigate to the Trip Details screen, the keyboard may automatically show, but you may like the keyboard lowered. You learned to do this in XML in *fragment_trip_list*, but in this case, you need to hide the keyboard *after* an edit view changes, as it does in *TripFragment*. Therefore, add the following to *onCreateView*.

```
1: getActivity().getWindow().setSoftInputMode(WindowManager.  
    LayoutParams.SOFT_INPUT_STATE_HIDDEN);
```

One of the most popular and useful sites for software developers is  stackoverflow.com, a question and answer site for programmers. When searching the site, supply as many keywords as you can to narrow your search results, and look for a green check mark next to a solution, indicating it is the accepted answer to the question.

- 34 (Optional) Have you tested with your device or emulator in the horizontal orientation? When you do, you will notice some of the views on Trip Details may not fit on the screen. Use  stackoverflow.com and search for “android xml lock orientation” to learn how to lock the orientation.
- 35 (Optional) Take a tour of  stackoverflow.com/tour and learn about many other high quality questions & answer sites at  stackexchange.com/sites.

CONCLUSION

1. Explain why a programmer might choose to always use the object versions of the integer and double primitive data types, and whether or not this would be a good decision.
 2. Use DeMorgan’s Law and *rewrite* your conditional statement that validates rating input. If you used OR, rewrite the statement as an AND; if you used AND, rewrite it using OR.
- Which conditional statement do you prefer and why?