Chapter 4
Flag Quiz Game App

Android How to Program, 3/e

Objectives

In this chapter you'll:

- Use `Fragments` to make better use of available screen real estate in an `Activity`'s GUI on phones and tablets.
- Display a settings icon on the app bar to enable users to access the app's user preferences.
- Automatically manage and persist an app's settings via a `PreferenceFragment`.
- Modify key–value pairs settings via a `SharedPreferences.Editor`.
- Organize image resources in the app's `assets` subfolders and manipulate them with an `AssetManager`.
- Define an animation and apply it to a `View`.
- Use a `Handler` to schedule a future task to perform on the GUI thread.
- Use `Toasts` to display messages briefly to the user.
- Launch a specific `Activity` with an explicit `Intent`.
- Use collections from the `java.util` package.
- Define layouts for multiple device orientations.
- Use Android's logging mechanism to log error messages.

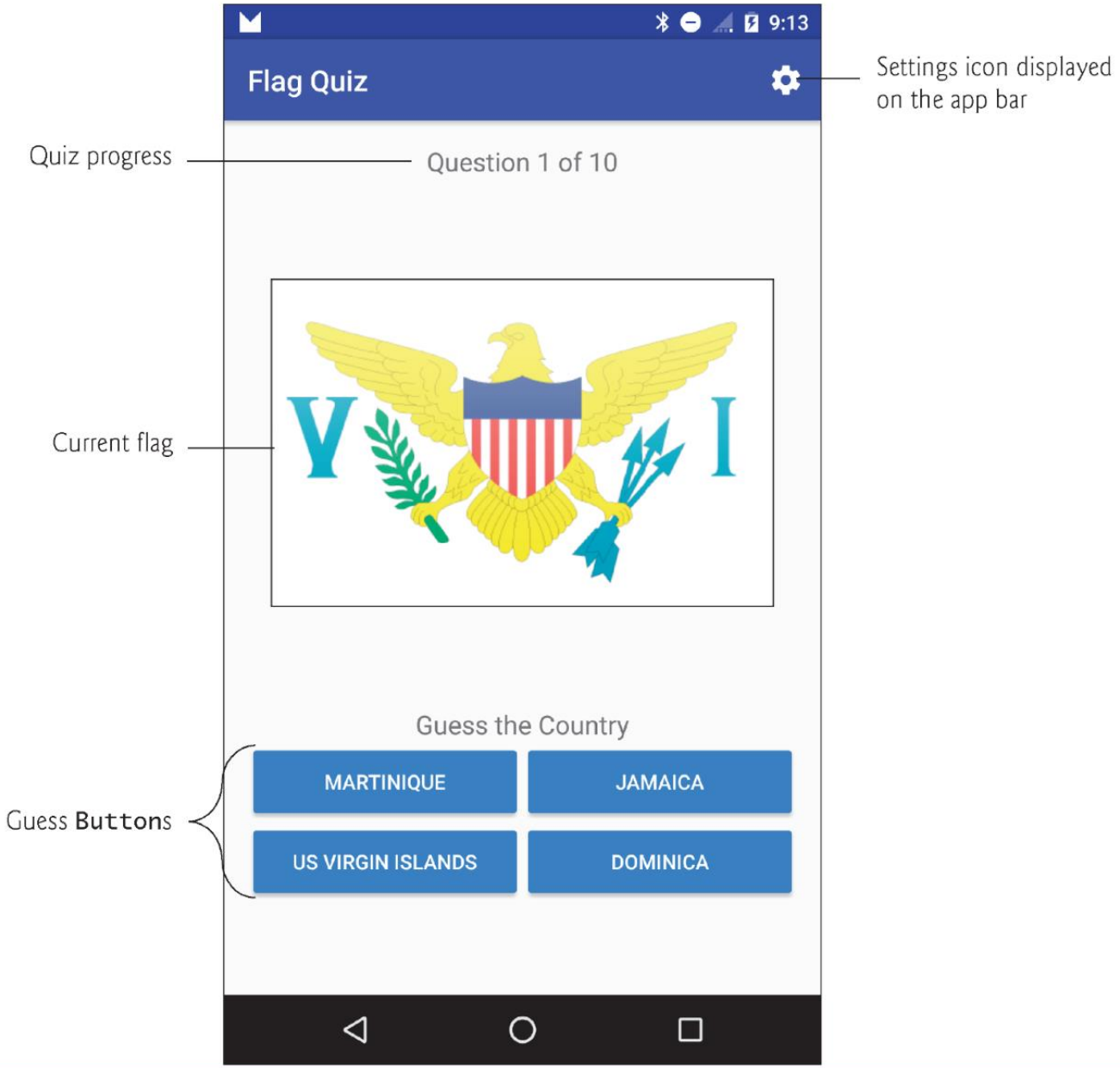

Fig. 4.1 | **Flag Quiz** app running on a smartphone in portrait orientation.

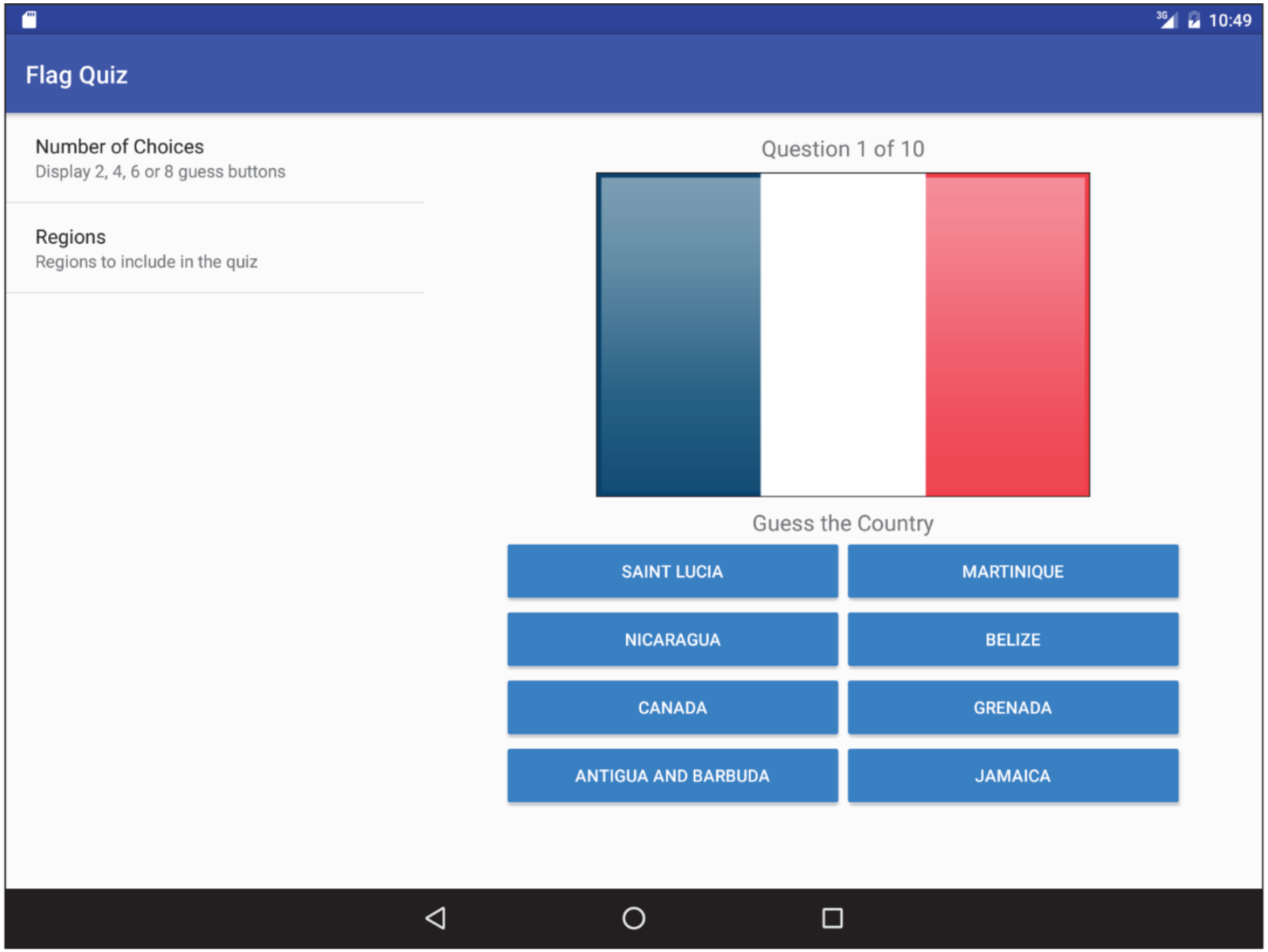

Fig. 4.2 | **Flag Quiz** app running on a tablet in landscape orientation.

a) Menu with the user touching **Number of Choices**

b) Dialog showing options for number of choices

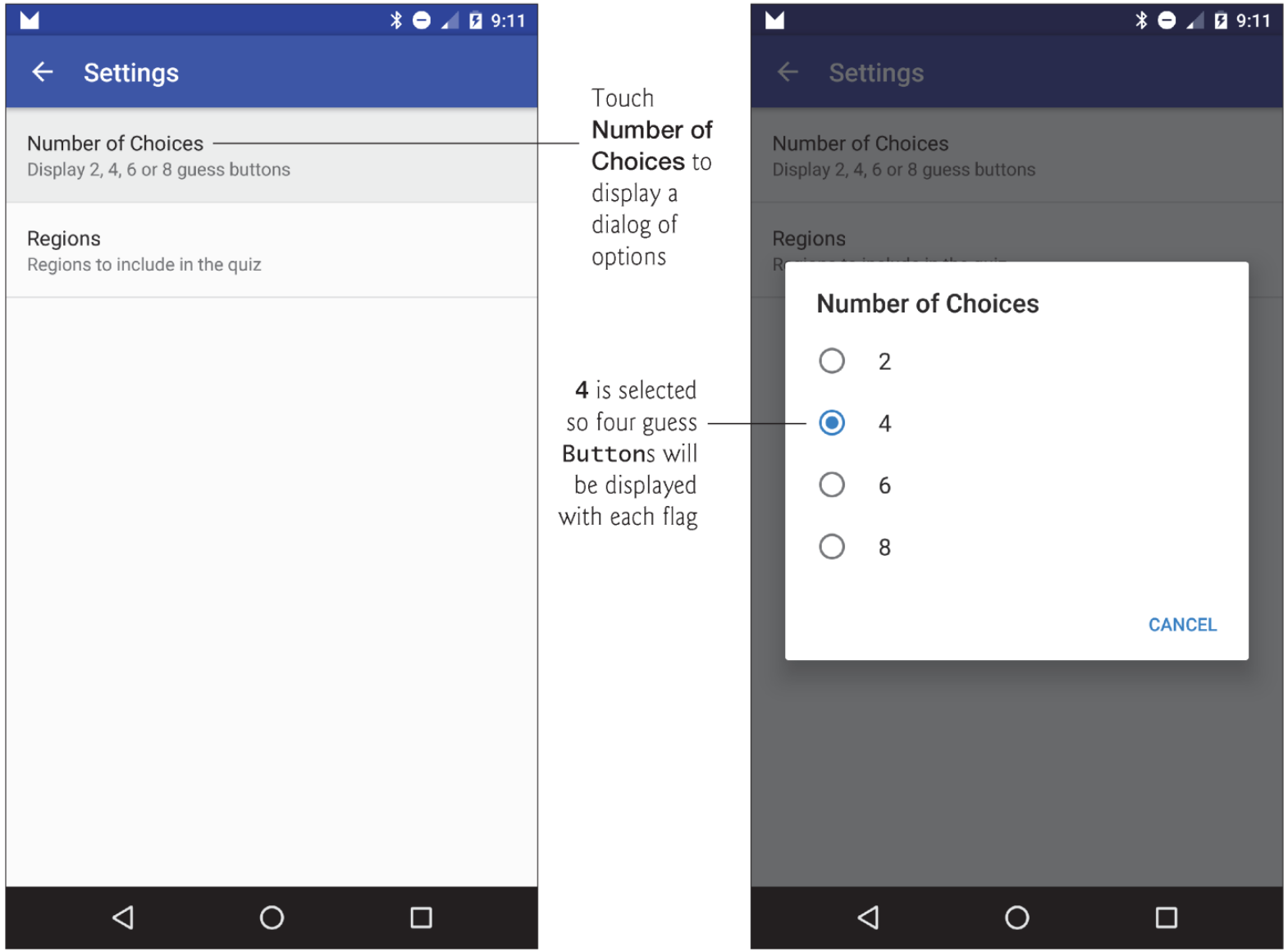

Fig. 4.3 | **Flag Quiz** settings screen and the **Number of Choices** dialog.

a) Menu with the user touching **Regions**

b) Dialog showing regions

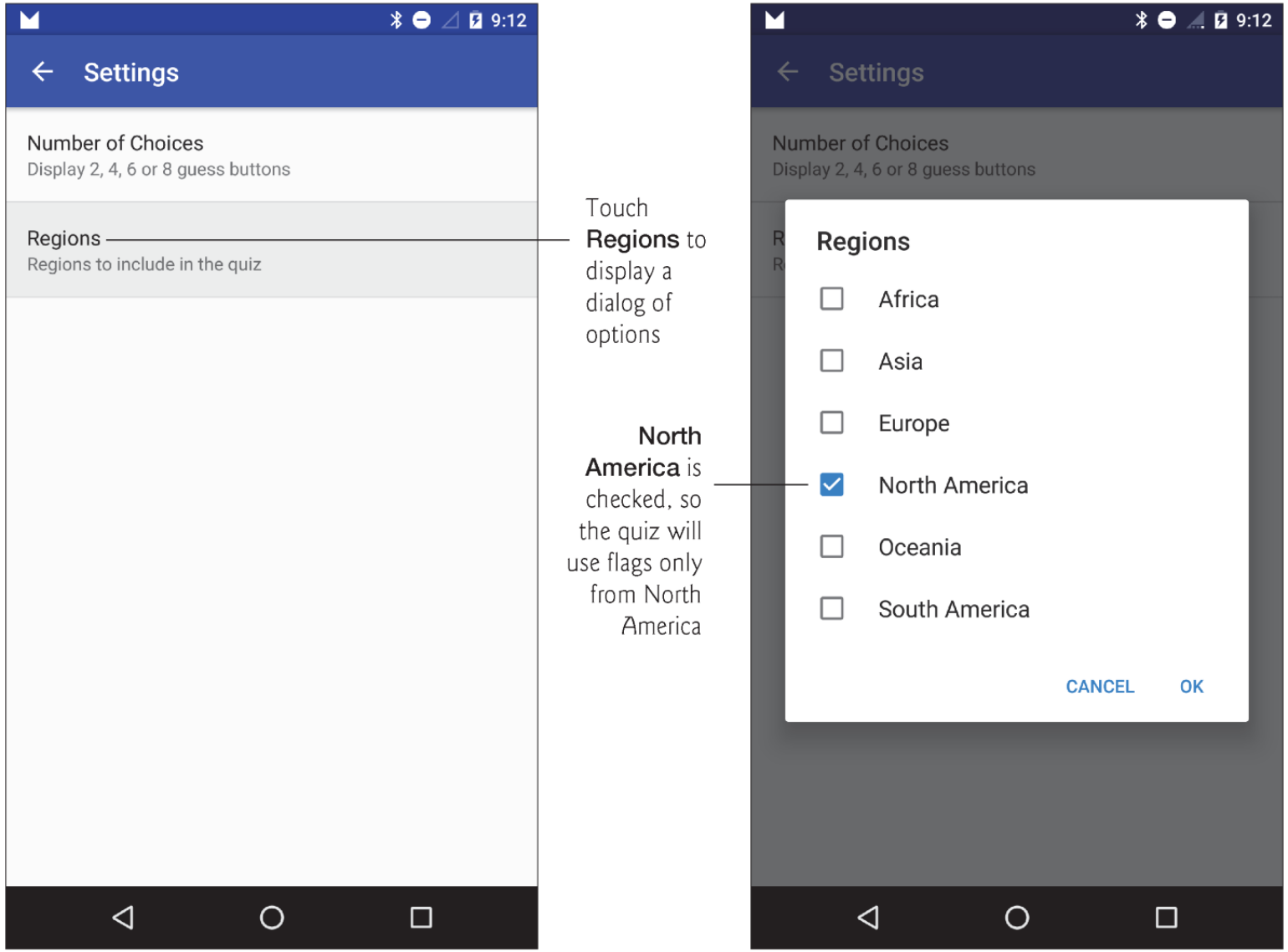

Fig. 4.4 | **Flag Quiz** settings screen and the **Regions** dialog (after unchecking **Africa**, **Asia**, **Europe**, **Oceania** and **South America**).

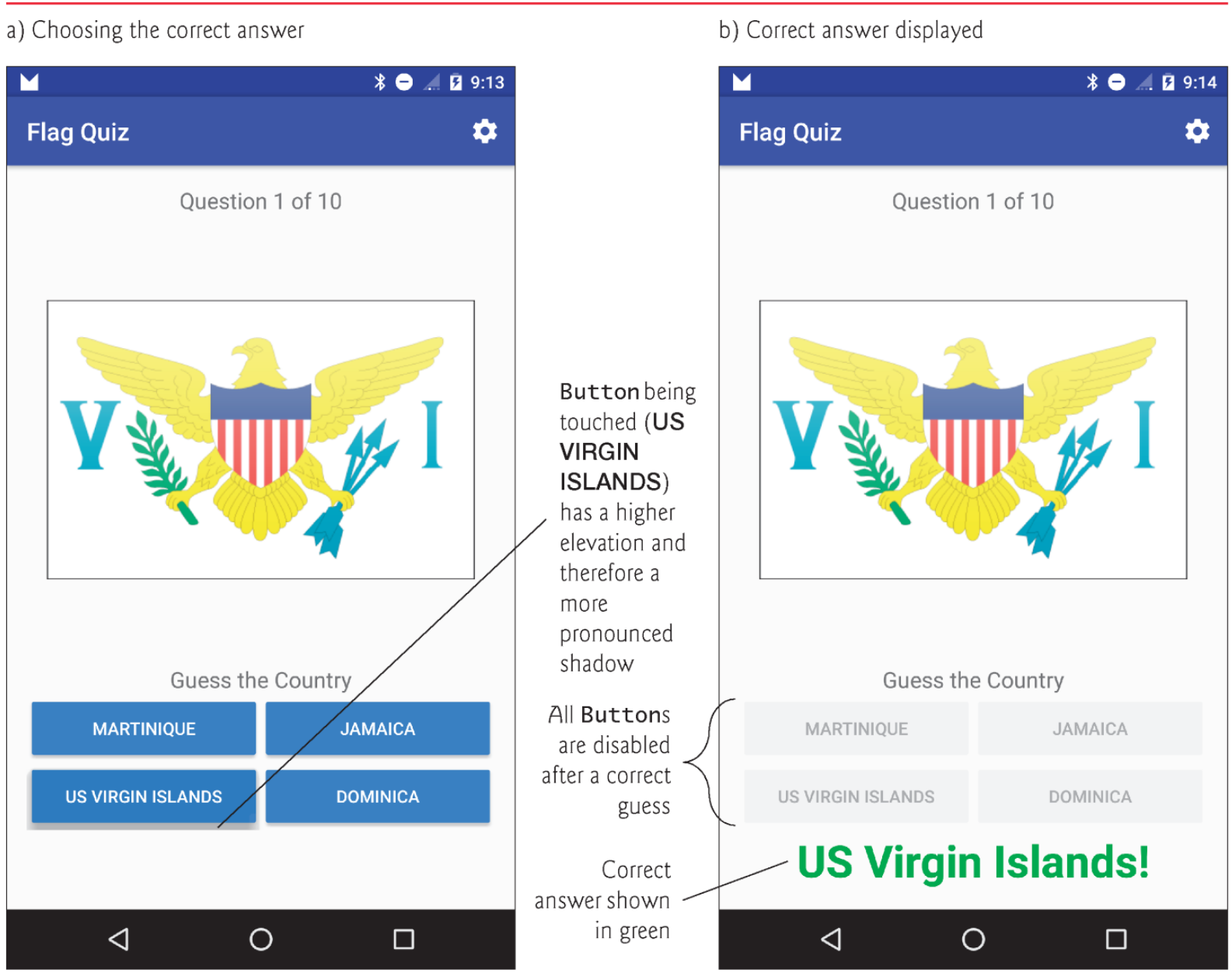

Fig. 4.5 | User choosing the correct answer and the correct answer displayed.

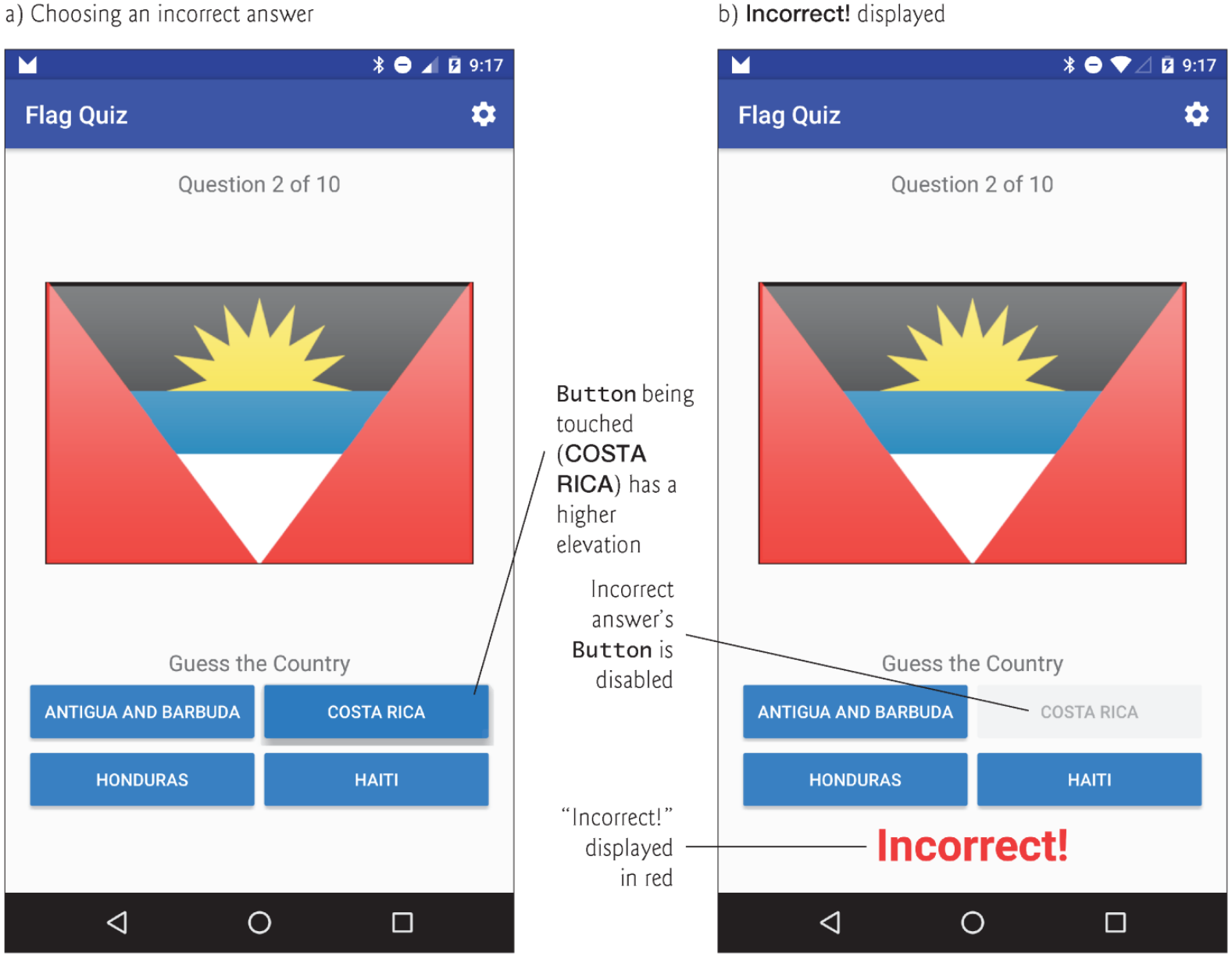

Fig. 4.6 | Disabled incorrect answer in the **Flag Quiz** app.

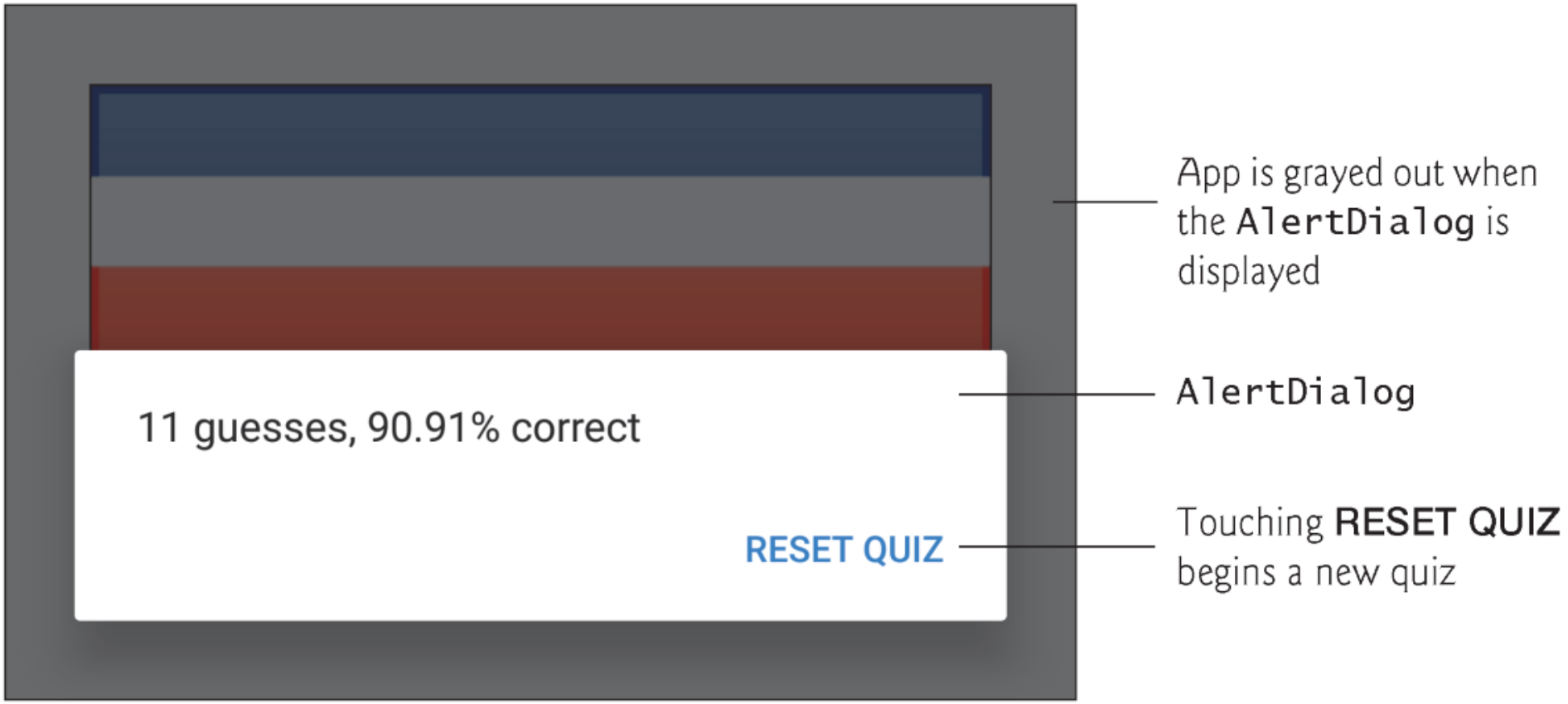

Fig. 4.7 | Results displayed after quiz completion.

Resource subfolder	Description
`anim`	Folder names that begin with `anim` contain XML files that define *tweened animations*, which can change an object's *transparency*, *size*, *position* and *rotation* over time. You'll define such an animation in Section , then in Section you'll play it to create a *shake effect* to provide the user with visual feedback for an incorrect guess.
`animator`	Folder names that begin with `animator` contain XML files that define *property animations*, which change the value of an object's property over time. In Java, a property is typically implemented in a class as an instance variable with both *set* and *get* accessors.
`color`	Folder names that begin with `color` contain XML files that define color state lists—lists of colors for various states, such as the states of a `Button` (*unpressed*, *pressed*, *enabled*, *disabled*, and so on). We'll use a color state list to define separate colors for when the guess `Button`s are *enabled* or *disabled* in Section .

Fig. 4.8 | Other subfolders within a project's `res` folder.

Resource subfolder	Description
`raw`	Folder names that begin with `raw` contain resource files (such as audio clips) that are read into an app as streams of bytes. We'll use such resources in Chapter 6 to play sounds.
`menu`	Folder names that begin with `menu` contain XML files that describe the contents of menus. When you create a project, the IDE automatically defines a menu with a **Settings** option.
`xml`	Folder names that begin with `xml` contain XML files that do not fit into the other resource categories—often XML data files used by the app. In Section , you'll create an XML file that represents the preferences displayed by this app's `SettingsActivityFragment`.

Fig. 4.8 | Other subfolders within a project's `res` folder.

Error-Prevention Tip 4.1

Operations that interact with or modify the GUI must be performed in the GUI thread (also called the UI thread or main thread), because GUI components are not thread safe.

Software Engineering Observation 4.1

Refer to collection objects using variables of the corresponding generic interface type, so you can change data structures easily without affecting the rest of your app's code.

Look-and-Feel Observation 4.1

The Android design guidelines indicate that text displayed in your GUI should be brief, simple and friendly with the important words first. For details on the recommended writing style, see `http://developer.android.com/design/style/writing.html`.

Key	Default value
number_of_choices_description	Display 2, 4, 6 or 8 guess buttons
world_regions	Regions
world_regions_description	Regions to include in the quiz
guess_country	Guess the Country
results	%1$d guesses, %2$.02f%% correct
incorrect_answer	Incorrect!
default_region_message	One region must be selected. Setting North America as the default region.
restarting_quiz	Quiz will restart with your new settings
question	Question %1$d of %2$d
reset_quiz	Reset Quiz
image_description	Image of the current flag in the quiz
default_region	North_America

Fig. 4.9 | `String` resources used in the **Flag Quiz** app.

Array resource name	Values
`regions_list`	`Africa, Asia, Europe, North_America, Oceania, South_America`
`regions_list_for_settings`	`Africa, Asia, Europe, North America, Oceania, South America`
`guesses_list`	`2, 4, 6, 8`

Fig. 4.10 | `String` array resources defined in `arrays.xml`.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

   <string-array name="regions_list">
      <item>Africa</item>
      <item>Asia</item>
      <item>Europe</item>
      <item>North_America</item>
      <item>Oceania</item>
      <item>South_America</item>
   </string-array>

```

Fig. 4.11 | `arrays.xml` defines `String` array resources used in the **Flag Quiz** app. (Part 1 of 2.)

```
13      <string-array name="regions_list_for_settings">
14         <item>Africa</item>
15         <item>Asia</item>
16         <item>Europe</item>
17         <item>North America</item>
18         <item>Oceania</item>
19         <item>South America</item>
20      </string-array>
21
22      <string-array name="guesses_list">
23         <item>2</item>
24         <item>4</item>
25         <item>6</item>
26         <item>8</item>
27      </string-array>
28
29   </resources>
```

Fig. 4.11 | `arrays.xml` defines `String` array resources used in the **Flag Quiz** app. (Part 2 of 2.)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3     <color name="colorPrimary">#3F51B5</color>
4     <color name="colorPrimaryDark">#303F9F</color>
5     <color name="colorAccent">#448AFF</color>
6     <color name="correct_answer">#00CC00</color>
7     <color name="incorrect_answer">#FF0000</color>
8  </resources>
```

Fig. 4.12 | `colors.xml` defines the app's color resources.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <selector xmlns:android="http://schemas.android.com/apk/res/android">
3     <item
4        android:color="@android:color/primary_text_dark"
5        android:state_enabled="true"/>
6
7     <item
8        android:color="@android:color/darker_gray"
9        android:state_enabled="false"/>
10 </selector>
```

Fig. 4.13 | `button_text_color.xml` defines a button's text color for the *enabled* and *disabled* states.

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <set xmlns:android="http://schemas.android.com/apk/res/android"
4     android:interpolator="@android:anim/decelerate_interpolator" >
5
6     <translate android:duration="100" android:fromXDelta="0"
7        android:toXDelta="-5%p" />
8
9     <translate android:duration="100" android:fromXDelta="-5%p"
10       android:toXDelta="5%p" android:startOffset="100" />
11
12    <translate android:duration="100" android:fromXDelta="5%p"
13       android:toXDelta="-5%p" android:startOffset="200" />
14 </set>
```

Fig. 4.14 | `incorrect_shake.xml` defines a flag animation that's played when the user makes an incorrect guess.

Property	Value	Description
`entries`	`@array/guesses_list`	An array of `String`s that will be displayed in the list of options.
`entryValues`	`@array/guesses_list`	An array of values associated with the options in the **Entries** property. The selected entry's value will be stored in the app's `SharedPreferences`.
`key`	`pref_numberOfChoices`	The name of the preference stored in the app's `SharedPreferences`.
`title`	`@string/number_of_choices`	The title of the preference displayed in the GUI.
`summary`	`@string/number_of_choices_description`	A summary description of the preference that's displayed below its title.
`persistent`	`true`	Whether the preference should persist after the app terminates—if `true`, class `PreferenceFragment` immediately persists the preference value each time it changes.
`defaultValue`	`4`	The item in the **Entries** property that's selected by default.

Fig. 4.15 | `ListPreference` property values.

Property	Value	Description
`entries`	`@array/regions_list_for_settings`	An array of `String`s that will be displayed in the list of options.
`entryValues`	`@array/regions_list`	An array of the values associated with the options in the **Entries** property. The selected entries' values will *all* be stored in the app's `SharedPreferences`.
`key`	`pref_regionsToInclude`	The name of the preference stored in the app's `SharedPreferences`.
`title`	`@string/world_regions`	The title of the preference displayed in the GUI.
`summary`	`@string/world_regions_description`	A summary description of the preference that's displayed below its title.
`persistent`	`true`	Whether the preference should persist after the app terminates.
`defaultValue`	`@array/regions_list`	An array of the default values for this preference—in this case, all of the regions will be selected by default.

Fig. 4.16 | `MultiSelectListPreference` property values.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <PreferenceScreen
3     xmlns:android="http://schemas.android.com/apk/res/android">
4
5     <ListPreference
6        android:entries="@array/guesses_list"
7        android:entryValues="@array/guesses_list"
8        android:key="pref_numberOfChoices"
9        android:title="@string/number_of_choices"
10       android:summary="@string/number_of_choices_description"
11       android:persistent="true"
12       android:defaultValue="4" />
13
14    <MultiSelectListPreference
15       android:entries="@array/regions_list_for_settings"
16       android:entryValues="@array/regions_list"
17       android:key="pref_regionsToInclude"
18       android:title="@string/world_regions"
19       android:summary="@string/world_regions_description"
20       android:persistent="true"
21       android:defaultValue="@array/regions_list" />
22
23 </PreferenceScreen>
```

Fig. 4.17 | `preferences.xml` defines the preferences displayed by the `SettingsActivityFragment`.

Look-and-Feel Observation 4.2

According to the Android design guidelines, `16dp` *is the recommended space between the edges of a device's touchable screen area and the app's content; however, many apps (such as games) use the full screen.*

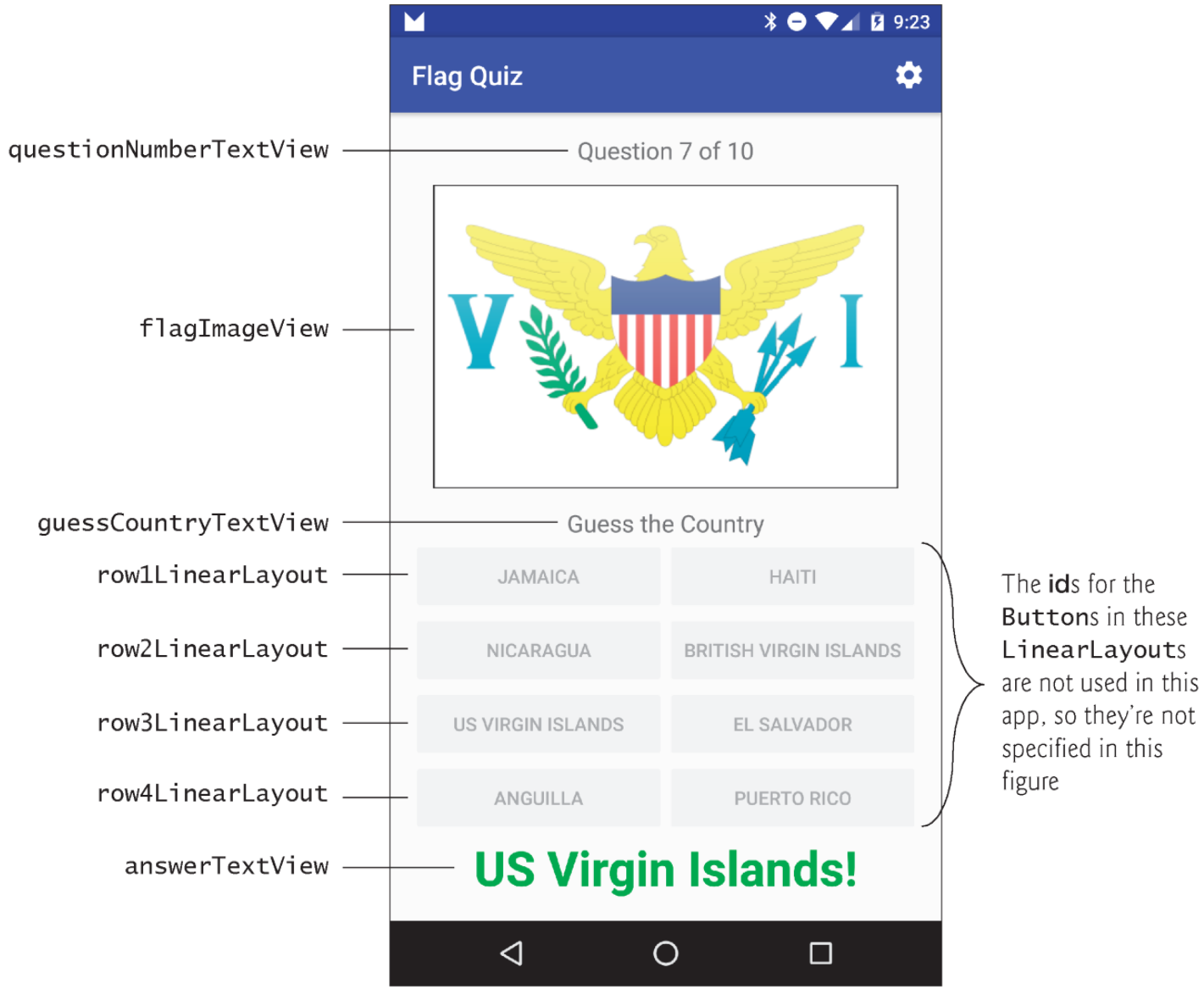

Fig. 4.18 | **Flag Quiz** GUI's components labeled with their **id** property values—the components are arranged in a vertical `LinearLayout`.

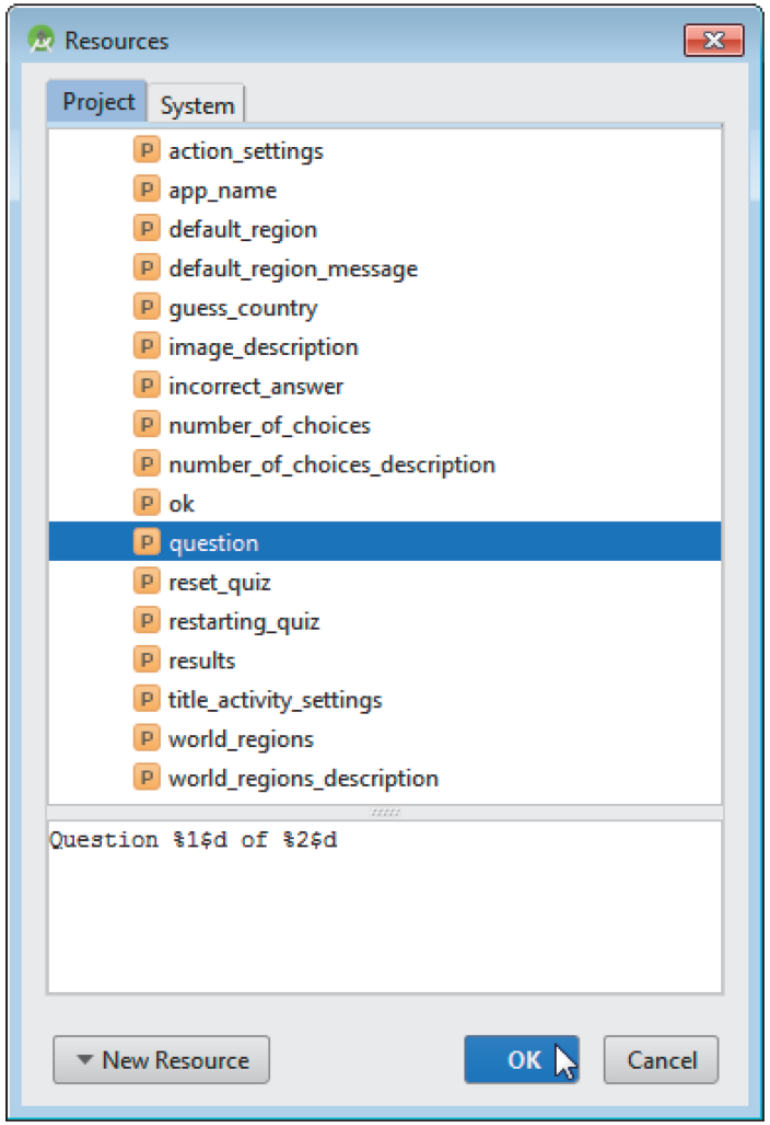

Fig. 4.19 | **Resource Chooser** dialog—selecting the existing `String` resource `question`.

Look-and-Feel Observation 4.3

Recall that it's considered a best practice in Android to ensure that every GUI component can be used with TalkBack. For components that don't have descriptive text, such as `ImageViews`*, set the component's* **contentDescription** *property.*

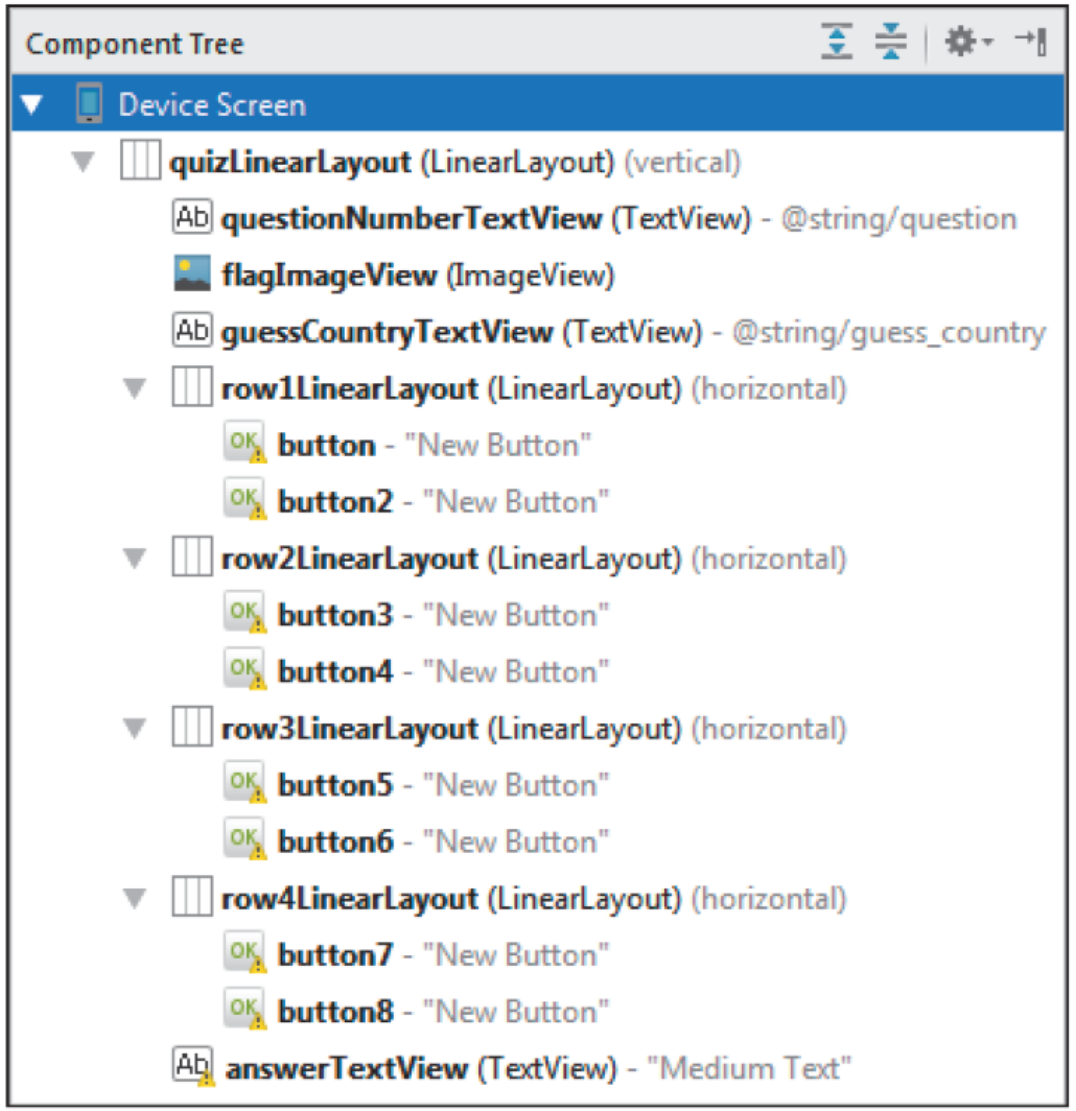

Fig. 4.20 | **Component Tree** window for `fragment_main.xml`.

GUI component	Property	Value
`Buttons`	*Layout Parameters*	
	layout:width	`0dp`
	layout:height	`match_parent`
	layout:weight	`1`
	Other Properties	
	lines	`2`
	textColor	`@color/button_text_color`
	style	`@android:style/Widget.Material.Button.Colored`

Fig. 4.21 | Property values for the `Buttons` components in `fragment_main.xml`.

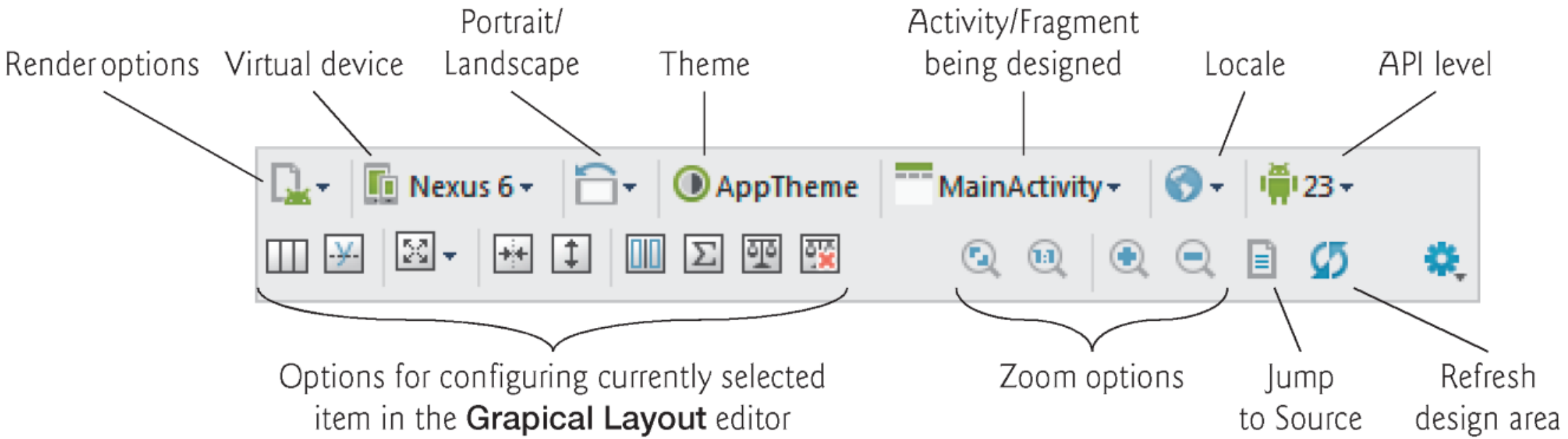

Fig. 4.22 | Canvas configuration options.

Option	Description
Render options	View one design screen at a time or see your design on a variety of screen sizes all at once.
Virtual device	Android runs on a wide variety of devices, so the layout editor comes with many device configurations that represent various screen sizes and resolutions that you can use to design your GUI. In this book, we use the predefined **Nexus 6** and **Nexus 9** screens, depending on the app. In Fig. 4.22, we selected **Nexus 6**.
Portrait/landscape	Toggles the design area between *portrait* and *landscape* orientations.
Theme	Can be used to set the theme for the GUI.
Activity/fragment being designed	Shows the `Activity` or `Fragment` class that corresponds to the GUI being designed.
Locale	For *internationalized* apps (Section), allows you to select a specific localization, so that you can see, for example, what your design looks like with different language strings.
API level	Specifies the target API level for the design. With each new API level, there have typically been new GUI features. The layout editor window shows only features that are available in the selected API level.

Fig. 4.23 | Explanation of the canvas configuration options.

```
1  // MainActivity.java
2  // Hosts the MainActivityFragment on a phone and both the
3  // MainActivityFragment and SettingsActivityFragment on a tablet
4  package com.deitel.flagquiz;
5
6  import android.content.Intent;
7  import android.content.SharedPreferences;
8  import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
9  import android.content.pm.ActivityInfo;
10 import android.content.res.Configuration;
11 import android.os.Bundle;
12 import android.preference.PreferenceManager;
13 import android.support.v7.app.AppCompatActivity;
14 import android.support.v7.widget.Toolbar;
15 import android.view.Menu;
16 import android.view.MenuItem;
17 import android.widget.Toast;
18
19 import java.util.Set;
20
```

Fig. 4.24 | MainActivity package statement and import statements.

```
21  public class MainActivity extends Activity {
22     // keys for reading data from SharedPreferences
23     public static final String CHOICES = "pref_numberOfChoices";
24     public static final String REGIONS = "pref_regionsToInclude";
25
26     private boolean phoneDevice = true; // used to force portrait mode
27     private boolean preferencesChanged = true; // did preferences change?
28
```

Fig. 4.25 | `MainActivity` declaration and fields.

```
29      // configure the MainActivity
30      @Override
31      protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.activity_main);
34         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
35         setSupportActionBar(toolbar);
36
37         // set default values in the app's SharedPreferences
38         PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
39
40         // register listener for SharedPreferences changes
41         PreferenceManager.getDefaultSharedPreferences(this).
42            registerOnSharedPreferenceChangeListener(
43               preferencesChangeListener);
44
```

Fig. 4.26 | `MainActivity` overridden `Activity` method `onCreate`. (Part 1 of 2.)

```
45          // determine screen size
46          int screenSize = getResources().getConfiguration().screenLayout &
47             Configuration.SCREENLAYOUT_SIZE_MASK;
48
49          // if device is a tablet, set phoneDevice to false
50          if (screenSize == Configuration.SCREENLAYOUT_SIZE_LARGE ||
51             screenSize == Configuration.SCREENLAYOUT_SIZE_XLARGE)
52             phoneDevice = false; // not a phone-sized device
53
54          // if running on phone-sized device, allow only portrait orientation
55          if (phoneDevice)
56             setRequestedOrientation(
57                ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
58       }
59
```

Fig. 4.26 | `MainActivity` overridden `Activity` method `onCreate`. (Part 2 of 2.)

```
60      // called after onCreate completes execution
61      @Override
62      protected void onStart() {
63         super.onStart();
64
65         if (preferencesChanged) {
66            // now that the default preferences have been set,
67            // initialize MainActivityFragment and start the quiz
68            MainActivityFragment quizFragment = (MainActivityFragment)
69               getSupportFragmentManager().findFragmentById(
70                  R.id.quizFragment);
71            quizFragment.updateGuessRows(
72               PreferenceManager.getDefaultSharedPreferences(this));
73            quizFragment.updateRegions(
74               PreferenceManager.getDefaultSharedPreferences(this));
75            quizFragment.resetQuiz();
76            preferencesChanged = false;
77         }
78      }
79
```

Fig. 4.27 | `MainActivity` overridden `Activity` method `onStart`.

```
80      // show menu if app is running on a phone or a portrait-oriented tablet
81      @Override
82      public boolean onCreateOptionsMenu(Menu menu) {
83         // get the device's current orientation
84         int orientation = getResources().getConfiguration().orientation;
85
86         // display the app's menu only in portrait orientation
87         if (orientation == Configuration.ORIENTATION_PORTRAIT) {
88            // inflate the menu
89            getMenuInflater().inflate(R.menu.menu_main, menu);
90            return true;
91         }
92         else
93            return false;
94      }
95
```

Fig. 4.28 | `MainActivity` overridden `Activity` method `onCreateOptionsMenu`.

```
96      // displays the SettingsActivity when running on a phone
97      @Override
98      public boolean onOptionsItemSelected(MenuItem item) {
99         Intent preferencesIntent = new Intent(this, SettingsActivity.class);
100        startActivity(preferencesIntent);
101        return super.onOptionsItemSelected(item);
102     }
103
```

Fig. 4.29 | `MainActivity` overridden `Activity` method `onOptionsItemSelected`.

```
104    // listener for changes to the app's SharedPreferences
105    private OnSharedPreferenceChangeListener preferencesChangeListener =
106       new OnSharedPreferenceChangeListener() {
107          // called when the user changes the app's preferences
108          @Override
109          public void onSharedPreferenceChanged(
110             SharedPreferences sharedPreferences, String key) {
111             preferencesChanged = true; // user changed app settings
112
113             MainActivityFragment quizFragment = (MainActivityFragment)
114                getSupportFragmentManager().findFragmentById(
115                   R.id.quizFragment);
116
```

Fig. 4.30 | Anonymous Inner class that implements `OnSharedPreferenceChangeListener`. (Part 1 of 3.)

```
117             if (key.equals(CHOICES)) { // # of choices to display changed
118                quizFragment.updateGuessRows(sharedPreferences);
119                quizFragment.resetQuiz();
120             }
121             else if (key.equals(REGIONS)) { // regions to include changed
122                Set<String> regions =
123                   sharedPreferences.getStringSet(REGIONS, null);
124
125                if (regions != null && regions.size() > 0) {
126                   quizFragment.updateRegions(sharedPreferences);
127                   quizFragment.resetQuiz();
128                }
```

Fig. 4.30 | Anonymous Inner class that implements `OnSharedPreferenceChangeListener`. (Part 2 of 3.)

```
129                 else {
130                    // must select one region--set North America as default
131                    SharedPreferences.Editor editor =
132                       sharedPreferences.edit();
133                    regions.add(getString(R.string.default_region));
134                    editor.putStringSet(REGIONS, regions);
135                    editor.apply();
136
137                    Toast.makeText(MainActivity.this,
138                       R.string.default_region_message,
139                       Toast.LENGTH_SHORT).show();
140                 }
141              }
142
143              Toast.makeText(MainActivity.this,
144                 R.string.restarting_quiz,
145                 Toast.LENGTH_SHORT).show();
146           }
147        };
148     }
```

Fig. 4.30 | Anonymous Inner class that implements `OnSharedPreferenceChangeListener`. (Part 3 of 3.)

Fig. 4.31 | `Toast` displayed after a preference is changed.

```
1  // MainActivityFragment.java
2  // Contains the Flag Quiz logic
3  package com.deitel.flagquiz;
4
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.security.SecureRandom;
8  import java.util.ArrayList;
9  import java.util.Collections;
10 import java.util.List;
11 import java.util.Set;
12
13 import android.animation.Animator;
14 import android.animation.AnimatorListenerAdapter;
15 import android.app.AlertDialog;
16 import android.app.Dialog;
17 import android.content.DialogInterface;
18 import android.content.SharedPreferences;
19 import android.content.res.AssetManager;
20 import android.graphics.drawable.Drawable;
```

Fig. 4.32 | `MainActivityFragment` `package` statement, `import` statements. (Part 1 of 2.)

```
21 import android.os.Bundle;
22 import android.support.v4.app.DialogFragment;
23 import android.support.v4.app.Fragment;
24 import android.os.Handler;
25 import android.util.Log;
26 import android.view.LayoutInflater;
27 import android.view.View;
28 import android.view.View.OnClickListener;
29 import android.view.ViewAnimationUtils;
30 import android.view.ViewGroup;
31 import android.view.animation.Animation;
32 import android.view.animation.AnimationUtils;
33 import android.widget.Button;
34 import android.widget.ImageView;
35 import android.widget.LinearLayout;
36 import android.widget.TextView;
37
```

Fig. 4.32 | `MainActivityFragment` `package` statement, `import` statements. (Part 2 of 2.)

```
38  public class MainActivityFragment extends Fragment {
39     // String used when logging error messages
40     private static final String TAG = "FlagQuiz Activity";
41
42     private static final int FLAGS_IN_QUIZ = 10;
43
44     private List<String> fileNameList; // flag file names
45     private List<String> quizCountriesList; // countries in current quiz
46     private Set<String> regionsSet; // world regions in current quiz
47     private String correctAnswer; // correct country for the current flag
48     private int totalGuesses; // number of guesses made
49     private int correctAnswers; // number of correct guesses
50     private int guessRows; // number of rows displaying guess Buttons
51     private SecureRandom random; // used to randomize the quiz
52     private Handler handler; // used to delay loading next flag
53     private Animation shakeAnimation; // animation for incorrect guess
54
55     private LinearLayout quizLinearLayout; // layout that contains the quiz
56     private TextView questionNumberTextView; // shows current question #
57     private ImageView flagImageView; // displays a flag
58     private LinearLayout[] guessLinearLayouts; // rows of answer Buttons
59     private TextView answerTextView; // displays correct answer
60
```

Fig. 4.33 | MainActivityFragment fields.

Good Programming Practice 4.1

For readability and modifiability, use `String` *constants to represent filenames* `String` *literals (such as those used as the names of files or to log error messages) that do not need to be localized, and thus are not defined in* `strings.xml`.

```
61      // configures the MainActivityFragment when its View is created
62      @Override
63      public View onCreateView(LayoutInflater inflater, ViewGroup container,
64         Bundle savedInstanceState) {
65         super.onCreateView(inflater, container, savedInstanceState);
66         View view =
67            inflater.inflate(R.layout.fragment_main, container, false);
68
69         fileNameList = new ArrayList<>();       // diamond operator
70         quizCountriesList = new ArrayList<>();
71         random = new SecureRandom();
72         handler = new Handler();
73
74         // load the shake animation that's used for incorrect answers
75         shakeAnimation = AnimationUtils.loadAnimation(getActivity(),
76            R.anim.incorrect_shake);
77         shakeAnimation.setRepeatCount(3); // animation repeats 3 times
78
```

Fig. 4.34 | `MainActivityFragment` overridden `Fragment` method `onCreateView`. (Part 1 of 3.)

```
79          // get references to GUI components
80          quizLinearLayout =
81             (LinearLayout) view.findViewById(R.id.quizLinearLayout);
82          questionNumberTextView =
83             (TextView) view.findViewById(R.id.questionNumberTextView);
84          flagImageView = (ImageView) view.findViewById(R.id.flagImageView);
85          guessLinearLayouts = new LinearLayout[4];
86          guessLinearLayouts[0] =
87             (LinearLayout) view.findViewById(R.id.row1LinearLayout);
88          guessLinearLayouts[1] =
89             (LinearLayout) view.findViewById(R.id.row2LinearLayout);
90          guessLinearLayouts[2] =
91             (LinearLayout) view.findViewById(R.id.row3LinearLayout);
92          guessLinearLayouts[3] =
93             (LinearLayout) view.findViewById(R.id.row4LinearLayout);
94          answerTextView = (TextView) view.findViewById(R.id.answerTextView);
95
```

Fig. 4.34 | `MainActivityFragment` overridden `Fragment` method `onCreateView`. (Part 2 of 3.)

```
96          // configure listeners for the guess Buttons
97          for (LinearLayout row : guessLinearLayouts) {
98             for (int column = 0; column < row.getChildCount(); column++) {
99                Button button = (Button) row.getChildAt(column);
100               button.setOnClickListener(guessButtonListener);
101            }
102         }
103
104         // set questionNumberTextView's text
105         questionNumberTextView.setText(
106            getString(R.string.question, 1, FLAGS_IN_QUIZ));
107         return view; // return the fragment's view for display
108      }
109
```

Fig. 4.34 | `MainActivityFragment` overridden `Fragment` method `onCreateView`. (Part 3 of 3.)

```
110     // update guessRows based on value in SharedPreferences
111     public void updateGuessRows(SharedPreferences sharedPreferences) {
112        // get the number of guess buttons that should be displayed
113        String choices =
114           sharedPreferences.getString(MainActivity.CHOICES, null);
115        guessRows = Integer.parseInt(choices) / 2;
116
117        // hide all guess button LinearLayouts
118        for (LinearLayout layout : guessLinearLayouts)
119           layout.setVisibility(View.GONE);
120
121        // display appropriate guess button LinearLayouts
122        for (int row = 0; row < guessRows; row++)
123           guessLinearLayouts[row].setVisibility(View.VISIBLE);
124     }
125
```

Fig. 4.35 | MainActivityFragment method updateGuessRows.

```
126     // update world regions for quiz based on values in SharedPreferences
127     public void updateRegions(SharedPreferences sharedPreferences) {
128        regionsSet =
129           sharedPreferences.getStringSet(MainActivity.REGIONS, null);
130     }
131
```

Fig. 4.36 | `MainActivityFragment` method `updateRegions`.

```
132     // set up and start the next quiz
133     public void resetQuiz() {
134        // use AssetManager to get image file names for enabled regions
135        AssetManager assets = getActivity().getAssets();
136        fileNameList.clear(); // empty list of image file names
137
138        try {
139           // loop through each region
140           for (String region : regionsSet) {
141              // get a list of all flag image files in this region
142              String[] paths = assets.list(region);
143
144              for (String path : paths)
145                 fileNameList.add(path.replace(".png", ""));
146           }
147        }
148        catch (IOException exception) {
149           Log.e(TAG, "Error loading image file names", exception);
150        }
```

Fig. 4.37 | `MainActivityFragment` method `resetQuiz`. (Part 1 of 2.)

```
151
152         correctAnswers = 0; // reset the number of correct answers made
153         totalGuesses = 0; // reset the total number of guesses the user made
154         quizCountriesList.clear(); // clear prior list of quiz countries
155
156         int flagCounter = 1;
157         int numberOfFlags = fileNameList.size();
158
159         // add FLAGS_IN_QUIZ random file names to the quizCountriesList
160         while (flagCounter <= FLAGS_IN_QUIZ) {
161            int randomIndex = random.nextInt(numberOfFlags);
162
163            // get the random file name
164            String filename = fileNameList.get(randomIndex);
165
166            // if the region is enabled and it hasn't already been chosen
167            if (!quizCountriesList.contains(filename)) {
168               quizCountriesList.add(filename); // add the file to the list
169               ++flagCounter;
170            }
171         }
172
173         loadNextFlag(); // start the quiz by loading the first flag
174      }
175
```

Fig. 4.37 | `MainActivityFragment` method `resetQuiz`. (Part 2 of 2.)

```
176     // after the user guesses a correct flag, load the next flag
177     private void loadNextFlag() {
178        // get file name of the next flag and remove it from the list
179        String nextImage = quizCountriesList.remove(0);
180        correctAnswer = nextImage; // update the correct answer
181        answerTextView.setText(""); // clear answerTextView
182
183        // display current question number
184        questionNumberTextView.setText(getString(
185           R.string.question, (correctAnswers + 1), FLAGS_IN_QUIZ));
186
187        // extract the region from the next image's name
188        String region = nextImage.substring(0, nextImage.indexOf('-'));
189
190        // use AssetManager to load next image from assets folder
191        AssetManager assets = getActivity().getAssets();
192
```

Fig. 4.38 | `MainActivityFragment` method `loadNextFlag`. (Part 1 of 4.)

```
193         // get an InputStream to the asset representing the next flag
194         // and try to use the InputStream
195         try (InputStream stream =
196            assets.open(region + "/" + nextImage + ".png")) {
197            // load the asset as a Drawable and display on the flagImageView
198            Drawable flag = Drawable.createFromStream(stream, nextImage);
199            flagImageView.setImageDrawable(flag);
200
201            animate(false); // animate the flag onto the screen
202         }
203         catch (IOException exception) {
204            Log.e(TAG, "Error loading " + nextImage, exception);
205         }
206
207         Collections.shuffle(fileNameList); // shuffle file names
208
209         // put the correct answer at the end of fileNameList
210         int correct = fileNameList.indexOf(correctAnswer);
211         fileNameList.add(fileNameList.remove(correct));
212
```

Fig. 4.38 | `MainActivityFragment` method `loadNextFlag`. (Part 2 of 4.)

```
213         // add 2, 4, 6 or 8 guess Buttons based on the value of guessRows
214         for (int row = 0; row < guessRows; row++) {
215            // place Buttons in currentTableRow
216            for (int column = 0;
217                 column < guessLinearLayouts[row].getChildCount();
218                 column++) {
219               // get reference to Button to configure
220               Button newGuessButton =
221                  (Button) guessLinearLayouts[row].getChildAt(column);
222               newGuessButton.setEnabled(true);
223
224               // get country name and set it as newGuessButton's text
225               String filename = fileNameList.get((row * 2) + column);
226               newGuessButton.setText(getCountryName(filename));
227            }
228         }
```

Fig. 4.38 | `MainActivityFragment` method `loadNextFlag`. (Part 3 of 4.)

```
229
230         // randomly replace one Button with the correct answer
231         int row = random.nextInt(guessRows); // pick random row
232         int column = random.nextInt(2); // pick random column
233         LinearLayout randomRow = guessLinearLayouts[row]; // get the row
234         String countryName = getCountryName(correctAnswer);
235         ((Button) randomRow.getChildAt(column)).setText(countryName);
236     }
237
```

Fig. 4.38 | `MainActivityFragment` method `loadNextFlag`. (Part 4 of 4.)

```
238     // parses the country flag file name and returns the country name
239     private String getCountryName(String name) {
240        return name.substring(name.indexOf('-') + 1).replace('_', ' ');
241     }
242
```

Fig. 4.39 | `MainActivityFragment` method `getCountryName`.

```
243     // animates the entire quizLinearLayout on or off screen
244     private void animate(boolean animateOut) {
245        // prevent animation into the the UI for the first flag
246        if (correctAnswers == 0)
247           return;
248
249        // calculate center x and center y
250        int centerX = (quizLinearLayout.getLeft() +
251           quizLinearLayout.getRight()) / 2;
252        int centerY = (quizLinearLayout.getTop() +
253           quizLinearLayout.getBottom()) / 2;
254
255        // calculate animation radius
256        int radius = Math.max(quizLinearLayout.getWidth(),
257           quizLinearLayout.getHeight());
258
259        Animator animator;
260
```

Fig. 4.40 | `MainActivityFragment` method `animate`. (Part 1 of 2.)

```
261         // if the quizLinearLayout should animate out rather than in
262         if (animateOut) {
263            // create circular reveal animation
264            animator = ViewAnimationUtils.createCircularReveal(
265               quizLinearLayout, centerX, centerY, radius, 0);
266            animator.addListener(
267               new AnimatorListenerAdapter() {
268                  // called when the animation finishes
269                  @Override
270                  public void onAnimationEnd(Animator animation) {
271                     loadNextFlag();
272                  }
273               }
274            );
275         }
276         else { // if the quizLinearLayout should animate in
277            animator = ViewAnimationUtils.createCircularReveal(
278               quizLinearLayout, centerX, centerY, 0, radius);
279         }
280
281         animator.setDuration(500); // set animation duration to 500 ms
282         animator.start(); // start the animation
283      }
284
```

Fig. 4.40 | `MainActivityFragment` method `animate`. (Part 2 of 2.)

```
285     // called when a guess Button is touched
286     private OnClickListener guessButtonListener = new OnClickListener() {
287        @Override
288        public void onClick(View v) {
289           Button guessButton = ((Button) v);
290           String guess = guessButton.getText().toString();
291           String answer = getCountryName(correctAnswer);
292           ++totalGuesses; // increment number of guesses the user has made
293
294           if (guess.equals(answer)) { // if the guess is correct
295              ++correctAnswers; // increment the number of correct answers
296
297              // display correct answer in green text
298              answerTextView.setText(answer + "!");
299              answerTextView.setTextColor(
300                 getResources().getColor(R.color.correct_answer,
301                    getContext().getTheme()));
302
303              disableButtons(); // disable all guess Buttons
304
```

Fig. 4.41 | Anonymous inner class that implements `OnClickListener`. (Part 1 of 4.)

```
305             // if the user has correctly identified FLAGS_IN_QUIZ flags
306             if (correctAnswers == FLAGS_IN_QUIZ) {
307                // DialogFragment to display quiz stats and start new quiz
308                DialogFragment quizResults =
309                   new DialogFragment() {
310                      // create an AlertDialog and return it
311                      @Override
312                      public Dialog onCreateDialog(Bundle bundle) {
313                         AlertDialog.Builder builder =
314                            new AlertDialog.Builder(getActivity());
315                         builder.setMessage(
316                            getString(R.string.results,
317                               totalGuesses,
318                               (1000 / (double) totalGuesses)));
319
```

Fig. 4.41 | Anonymous inner class that implements `OnClickListener`. (Part 2 of 4.)

```
320                 // "Reset Quiz" Button
321                 builder.setPositiveButton(R.string.reset_quiz,
322                    new DialogInterface.OnClickListener() {
323                       public void onClick(DialogInterface dialog,
324                          int id) {
325                          resetQuiz();
326                       }
327                    }
328                 );
329
330                 return builder.create(); // return the AlertDialog
331              }
332           };
333
334           // use FragmentManager to display the DialogFragment
335           quizResults.setCancelable(false);
336           quizResults.show(getFragmentManager(), "quiz results");
337        }
```

Fig. 4.41 | Anonymous inner class that implements `OnClickListener`. (Part 3 of 4.)

```
338                 else { // answer is correct but quiz is not over
339                    // load the next flag after a 2-second delay
340                    handler.postDelayed(
341                       new Runnable() {
342                          @Override
343                          public void run() {
344                             animate(true); // animate the flag off the screen
345                          }
346                       }, 2000); // 2000 milliseconds for 2-second delay
347                 }
348              }
349              else { // answer was incorrect
350                 flagImageView.startAnimation(shakeAnimation); // play shake
351
352                 // display "Incorrect!" in red
353                 answerTextView.setText(R.string.incorrect_answer);
354                 answerTextView.setTextColor(getResources().getColor(
355                    R.color.incorrect_answer, getContext().getTheme()));
356                 guessButton.setEnabled(false); // disable incorrect answer
357              }
358           }
359        };
360
```

Fig. 4.41 | Anonymous inner class that implements `OnClickListener`. (Part 4 of 4.)

Look-and-Feel Observation 4.4

You can set an `AlertDialog`*'s title (which appears above the dialog's message) with* `AlertDialog.Builder` *method* `setTitle`*. According to the Android design guidelines for dialogs (*`http://developer.android.com/design/building-blocks/dialogs.html`*), most dialogs do not need titles. A dialog should display a title for "a high-risk operation involving potential loss of data, connectivity, extra charges, and so on." Also, dialogs that display lists of options use the title to specify the dialog's purpose.*

```
361     // utility method that disables all answer Buttons
362     private void disableButtons() {
363        for (int row = 0; row < guessRows; row++) {
364           LinearLayout guessRow = guessLinearLayouts[row];
365           for (int i = 0; i < guessRow.getChildCount(); i++)
366              guessRow.getChildAt(i).setEnabled(false);
367        }
368     }
369  }
```

Fig. 4.42 | `MainActivityFragment` method `disableButtons`.

```
1  // SettingsActivity.java
2  // Activity to display SettingsActivityFragment on a phone
3  package com.deitel.flagquiz;
4
5  import android.os.Bundle;
6  import android.support.v7.app.AppCompatActivity;
7  import android.support.v7.widget.Toolbar;
8
9  public class SettingsActivity extends AppCompatActivity {
10    // inflates the GUI, displays Toolbar and adds "up" button
11    @Override
12    protected void onCreate(Bundle savedInstanceState) {
13       super.onCreate(savedInstanceState);
14       setContentView(R.layout.activity_settings);
15       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
16       setSupportActionBar(toolbar);
17       getSupportActionBar().setDisplayHomeAsUpEnabled(true);
18    }
19 }
```

Fig. 4.43 | `SettingsActivity` displays the `SettingsActivityFragment` on a phone device and on a tablet device in portrait orientation.

```
1  // SettingsActivityFragment.java
2  // Subclass of PreferenceFragment for managing app settings
3  package com.deitel.flagquiz;
4
5  import android.os.Bundle;
6  import android.preference.PreferenceFragment;
7
8  public class SettingsActivityFragment extends PreferenceFragment {
9     // creates preferences GUI from preferences.xml file in res/xml
10    @Override
11    public void onCreate(Bundle bundle) {
12       super.onCreate(bundle);
13       addPreferencesFromResource(R.xml.preferences); // load from XML
14    }
15 }
```

Fig. 4.44 | `SettingsActivityFragment` subclass of `PreferenceFragment` displays the app's preferences.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest package="com.deitel.flagquiz"
3     xmlns:android="http://schemas.android.com/apk/res/android">
4
5     <application
6        android:allowBackup="true"
7        android:icon="@mipmap/ic_launcher"
8        android:label="@string/app_name"
9        android:supportsRtl="true"
10       android:theme="@style/AppTheme">
11       <activity
12          android:name=".MainActivity"
13          android:label="@string/app_name"
14          android:launchMode="singleTop"
15          android:theme="@style/AppTheme.NoActionBar">
16          <intent-filter>
17             <action android:name="android.intent.action.MAIN"/>
18
19             <category android:name="android.intent.category.LAUNCHER"/>
20          </intent-filter>
```

Fig. 4.45 | `AndroidManifest.xml` with `SettingsActivity` declared. (Part 1 of 2.)

```
21          </activity>
22          <activity
23             android:name=".SettingsActivity"
24             android:label="@string/title_activity_settings"
25             android:parentActivityName=".MainActivity"
26             android:theme="@style/AppTheme.NoActionBar">
27             <meta-data
28                android:name="android.support.PARENT_ACTIVITY"
29                android:value="com.deitel.flagquiz.MainActivity">
30          </activity>
31       </application>
32
33    </manifest>
```

Fig. 4.45 | `AndroidManifest.xml` with `SettingsActivity` declared. (Part 2 of 2.)