Chapter 4: Explore!
Decision-Making Controls
Objectives

In this chapter, you learn to:

• Create an Android project with a custom icon
• Change the text color in controls using hexadecimal colors
• Align controls using the Change Gravity tool
• Determine layout with the Change Margins tool
• Place a RadioGroup and RadioButtons in Android applications
• Write code for a RadioGroup control
Objectives (continued)

• Make decisions using an If statement
• Make decisions using an If Else statement
• Make decisions using logical operators
• Display an Android toast notification
• Test the isChecked property
• Make decisions using nested if statements
The Medical Calculator App

• We will be creating an app to convert pounds to kilograms and kilograms to pounds
  – Formulas needed:
    • Kilograms = pounds * 2.2
    • Pounds = kilograms / 2.2
• App is designed to be used in a hospital setting to administer medication to patients based on patient weight
  – Hospital scales register pounds
  – Meds (based on patient weight) dispensed in kilograms
The Medical Calculator App (cont’d)

Figure 4-1 Opening Screen of the Medical Calculator

Figure 4-2 Results Screen of the Medical Calculator
The Medical Calculator App (cont’d)

• Steps to complete the App
  1. Create a customized launcher icon.
  2. Define a TextField for the data entry of the weight of the patient.
  3. Define a RadioGroup to select pounds to kilograms or kilograms to pounds.
  4. Display a Toast message for data validation.
  5. Convert data so it can be used for arithmetic operations.
  6. Perform arithmetic operations on data the user enters.
  7. Display formatted results.
The Launcher Icon

- **The Launcher Icon** allows you to view which apps are available
  - High-quality launcher icons can influence users to purchase your app
  - Icons can establish brand identity
  - Simple images with clear visual cues have a memorable impact
  - Icon dimensions are 72 X 72 pixels for the high-density screen
  - Vector graphics are best for icon design because images are easily resized
When you publish an app to the Android Market, you must provide a 512 x 512 pixel, high-resolution application icon in the developer console as you upload your program. This icon is displayed in the Android Market to provide a description of the app and does not replace your launcher icon.

Table 4-1 Launcher icon sizes

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Dots per Inch (dpi)</th>
<th>Size (px)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldpi (low-density screen)</td>
<td>120</td>
<td>36 x 36</td>
</tr>
<tr>
<td>mdpi (medium-density screen)</td>
<td>160</td>
<td>48 x 48</td>
</tr>
<tr>
<td>hdpi (high-density screen)</td>
<td>240</td>
<td>72 x 72</td>
</tr>
<tr>
<td>xhdpi (extra high-density screen)*</td>
<td>320</td>
<td>96 x 96</td>
</tr>
</tbody>
</table>

* Used by some tablets
The Launcher Icon (continued)

- Customizing a Launcher Icon
  - First, create a new project

Figure 4-5 Theme with action bar
The Launcher Icon (continued)

- Customizing a Launcher Icon (continued)
  - Next, drag the icon image into the drawable-hdpi folder
  - Then, modify the code in the Android Manifest file
    - Click in the line `android:icon="drawable/ic_launcher"`
    - Change the filename portion from `ic_launcher"` to `ic_launcher_weight"`. 
The Launcher Icon (continued)

• **Customizing a Launcher Icon (continued)**
  – Finally, add the selected theme to the Android Manifest file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.androidbootcamp.medicalcalculator"
    android:versionCode="1"
    android:versionName="1.0">

  <application>
    <activity>
      android:label="@string/app_name"
      android:theme="@android:style/Theme.WithActionBar"
    </activity>
  </application>

  <uses-sdk android:minSdkVersion="14"/>
</manifest>
```

Figure 4-8 Android manifest code with new theme
RadioButton and RadioGroup Controls

• **RadioButton** control selects or deselects an option
  – Can be arranged horizontally or vertically
  – Have a label defined by the text property
  – Can be initially set to checked or unchecked
  – Typically used together in a **RadioGroup**
    • Only one RadioButton in the group can be selected at a time
  – Good to offer a default selection (checked = true) for the option that is used most
RadioButton and RadioGroup Controls (cont’d)

• Changing the Text Color of Android Controls
  – Use hexadecimal color codes to represent RGB (Red, Green, Blue) values
  – Codes range from 00 to FF (00 = none, FF = full)
  – Codes are identified by a pound sign, followed by the RGB values
    • #FF0000 is all RED
    • #00FF00 is all GREEN
    • #0000FF is all BLUE
    • #FFFF00 is YELLOW (RED and GREEN = YELLOW)
RadioButton and RadioGroup Controls (cont’d)

• Changing the Layout Gravity
  – Linear layout is the default setting on the emulator
  – The Change Gravity tool changes the alignment
    • Works like the left, center, right, top or bottom buttons on the Microsoft Office ribbon
RadioButton and RadioGroup Controls (cont’d)

Figure 4-9 Change Gravity tool
Changing the Margins

- **Margins** allow for more flexibility in controlling your layout
- Set independent pixel values instead of “eyeballing” to create equal spaces around controls
- Using the same specified margins creates a symmetrical layout
RadioButton and RadioGroup Controls  (cont’d)

• Adding the RadioButton Group
  – Use the prefix rad (radLbToKilo) to name the control
RadioButton and RadioGroup Controls (cont’d)

• Coding a RadioButton Control

```java
final RadioButton lbsToKilo = (RadioButton) findViewById(R.id.radLbToKilo);
final RadioButton kiloToLbs = (RadioButton) findViewById(R.id.radKiloToLb);
```

Figure 4-15 EditText and RadioButtons referenced
Making Decisions with Conditional Statements

- **Decision structures** are used to test conditions

- **Using an If Statement**
  
  ```java
  If (condition) {
      // Statements completed if condition is true
  }
  ```

  - Statements between the opening and closing braces are executed if the condition is true
• Using **If Else Statements**

If (condition) {
    // Statements completed if condition is true
} else {
    // Statements completed if condition is false
}

– One set of statements are executed if the condition is true and a different set of statements are executed if the condition is false
Making Decisions with Conditional Statements (continued)

- Relational Operators
  - Java strings are compared with the equals method (==) of the string class

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Resulting Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
<td>6 == 6</td>
<td>True</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>4 != 7</td>
<td>False</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>3 &gt; 2</td>
<td>True</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>8 &lt; 1</td>
<td>False</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>5 &gt;= 5</td>
<td>True</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>9 &lt;= 6</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 4-2 Relational operators
– Use the `compareTo` method to check if two strings are equal

<table>
<thead>
<tr>
<th>If Statement</th>
<th>Comparison</th>
<th>Resulting Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (name1.equals(name2))</td>
<td>Strings are not equal</td>
<td>False</td>
</tr>
<tr>
<td>if (name1.compareTo(name1) == 0)</td>
<td>Strings are equal</td>
<td>True</td>
</tr>
<tr>
<td>if (name1.compareTo(name3) == 0)</td>
<td>Strings are not equal</td>
<td>False</td>
</tr>
<tr>
<td>if (name1.compareTo(name2) &gt; 0)</td>
<td>The first string precedes the second string; returns a negative number</td>
<td>False</td>
</tr>
<tr>
<td>if (name1.compareTo(name3) &lt; 0)</td>
<td>The first string follows the third string; returns a negative number</td>
<td>True</td>
</tr>
<tr>
<td>if (name3.compareTo(name2) &gt; 0)</td>
<td>The first string follows the second string; returns a positive number</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 4-3 Examples of the `equals` and `compareTo` methods
Logical Operators

When more than one condition is tested the conditions are called a compound condition.

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>And—all conditions must be true</td>
<td>if (flight &lt; 400 &amp;&amp; hotel &lt; 120)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Not—reverses the meaning of a condition</td>
<td>if (! (grade &gt; 70))</td>
</tr>
</tbody>
</table>

Table 4-4 Common logical operators
Making Decisions with Conditional Statements (continued)

- Data Validation
  - User entries must be checked for reasonable values
- Toast Notification
  - A toast notification communicates messages to the user (message slides upward into view like toast popping out of a toaster)
  - Uses a Toast Object and the MakeText() method

```java
Toast.makeText(Main.this, "Pounds must be less than 500", Toast.LENGTH_LONG).show();
```
Using the `isChecked()` Method of RadioButton Controls

- The `isChecked()` method determines if the RadioButton object has been selected

```java
if (lbToKilo.isChecked) {
    // statements completed if condition is true
} else {
    // statements completed if condition is false
}
```
• Nested If Statements
  – If statements are **nested** when one if statement is inside of another if statement

```java
if (lbToKilo.isChecked) {
    if (weightEntered <= 500) {
        convertedWeight = weightEntered / conversionRate;
    } else {
        Toast.makeText(Main.this, "Pounds must be less than 500", Toast.LENGTH_LONG).show();
    }
}
```
• Coding the Button Event

```java
convert.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        weightEntered=Double.parseDouble(weight.getText().toString());
        DecimalFormat tenth = new DecimalFormat("#.#");
    }
});
```

Figure 4-19 Rounding off a number

DecimalFormat rounds off to one place past the decimal point
Making Decisions with Conditional Statements (continued)

• Coding the nested If Statements

```java
public void onClick(View v) {
    // TODO Auto-generated method stub
    weightEntered = Double.parseDouble(weight.getText().toString());
    DecimalFormat tenth = new DecimalFormat("#.#");
    if (lbToKilo.isChecked()) {
        if (weightEntered <= 500) {
            convertedWeight = weightEntered / conversionRate;
            result.setText(tenth.format(convertedWeight) + " kilograms");
        } else {
            Toast.makeText(Main.this, "Pounds must be less than 500", Toast.LENGTH_LONG).show();
        }
    }
}
```
Making Decisions with Conditional Statements (continued)

Figure 4-24 Completed code
Summary

- To display a custom launcher icon, copy the custom image to the res/drawable folder and update the Android Manifest file
- Include RadioButton controls to allow users to select or deselect options – only one button can be selected at a time
- Android apps use hexadecimal color codes
- Use the layout gravity property to position a control precisely on the screen; use change margins to change spacing between objects
• If statements execute statements if a condition is true
• If Else statements execute one group of statements if a condition is true and different group of statements if the condition is false
• Relational operators are used within the conditional statement
• Compound conditions must use logical operators such as && (And)
Summary (continued)

- Toast notifications display a brief message to a user
- Use nested If statements to test a second condition only after determining that a first condition is true or false
- Statements are nested when one If statement is within another If statement