Exploring Building Blocks

Based on Chapter 7, but expanded by KJK

Overview

- Understand Android views, controls, and layouts
- Accept input with EditText
- Display text to users with TextView
- Give users choices using Spinner controls
- Allow simple user selections with buttons, check boxes, switches, and radio groups
- Retrieve dates and times from users
- Use indicators to display data to users
- Play a movie

The Android Controls

- The Android SDK contains a Java package named `android.widget`.
- An Android control typically refers to a class within this package.
- The Android SDK includes classes to representing most common UI controls, including:
  - `ImageView`
  - `FrameLayout`
  - `EditText`
  - `Button`
- As mentioned previously, all controls are typically derived from the `View` class.

The Android Controls

- Each control you want to be able to access programmatically must have a unique identifier specified using the `android:id` attribute.
  - You use this identifier to access the control with the `findViewById()` method in your Activity class, e.g.,

```
TextView view = findViewById( R.id.textView );
```

Retrieving Data from Users with EditText

- The Android SDK provides a number of controls for retrieving data from users.
- One of the most common types of data that applications often need to collect from users is text.
- Two frequently used views to handle this type of job are `EditText` and `Spinner` controls.

Retrieving Text Input Using EditText Controls

- The Android SDK provides an `EditText` to handle text input from a user (it is similar to a `TextField` in JavaFX).
- The `EditText` class is derived from `TextView`.
- Most of its functionality is contained within `TextView` but is enabled when created as an `EditText`.

Retrieving Text Input Using EditText Controls

```
<EditText
    android:id="@+id/editText01"
    android:layout_height="wrap_content"
    android:hint=”Enter name"
    android:lines="4"
    android:layout_width=”match_parent" />
```

We will start using XML notation, even though we will mostly use the Layout Editor.

Retrieving Text Input Using EditText Controls

Elemen name (tag)

```
<EditText
    android:id="@+id/editText01"
    android:layout_height="wrap_content"
    android:hint=”Enter name"
    android:lines="4"
    android:layout_width=”match_parent" />
```

Atrtribute name

Atrtribute value

We will start using XML notation, even though we will mostly use the Layout Editor.

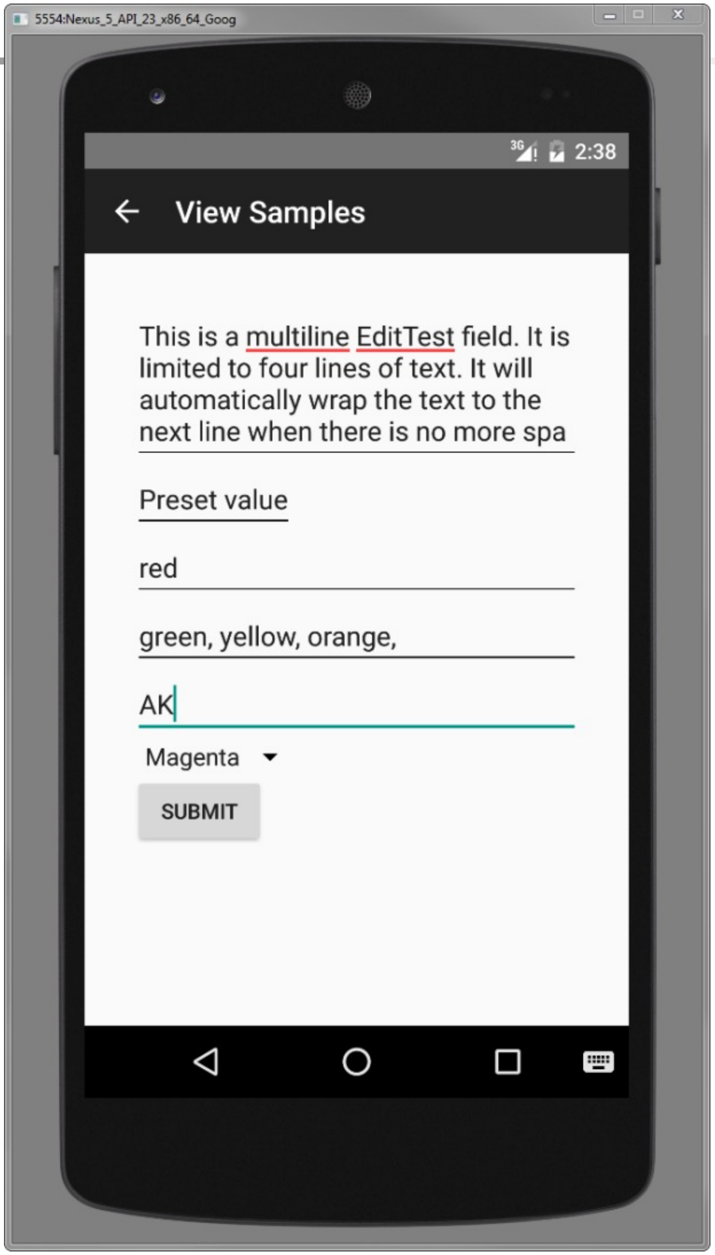
Retrieving Text Input Using EditText Controls

- Reading data from EditText:

```
EditText nameET = findViewById( R.id.editText01 );
String name = nameET.getText().toString();
```

- `nameET.getText()` returns an `Editable` type and `toString` is necessary.

Retrieving Text Input Using EditText Controls



This is a multiline EditTest field. It is limited to four lines of text. It will automatically wrap the text to the next line when there is no more spa

Preset value

red

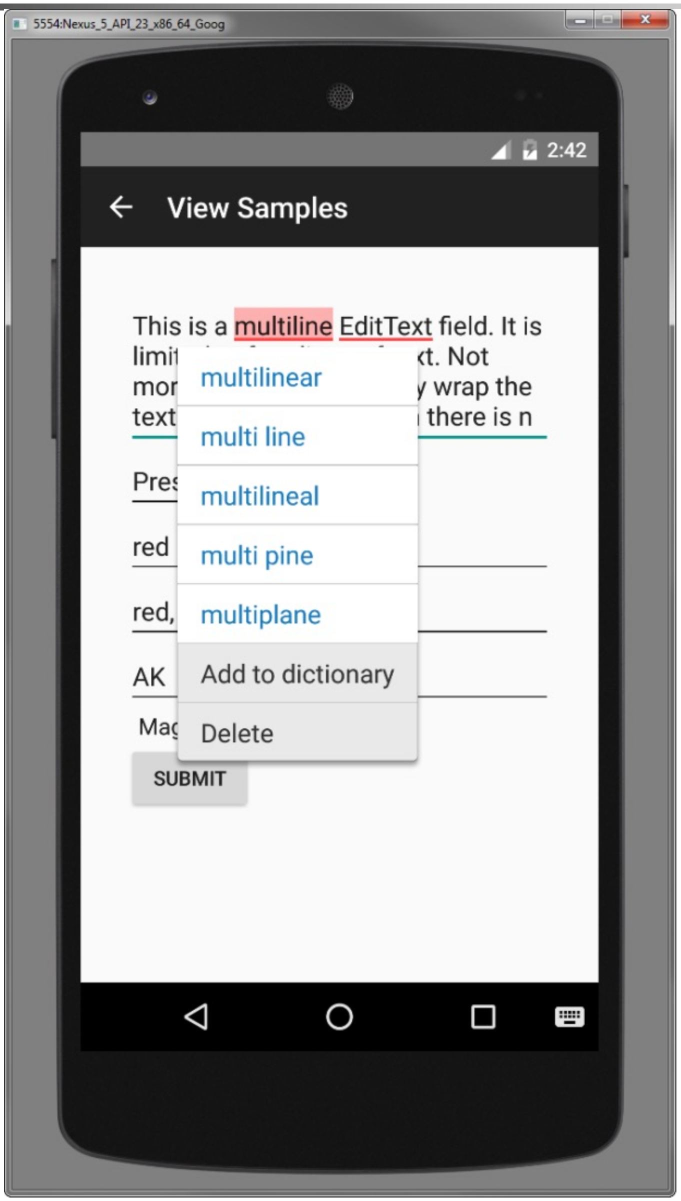
green, yellow, orange,

AK

Magenta

SUBMIT

Retrieving Text Input Using EditText Controls



This is a multiline EditText field. It is
limi xt. Not
mor y wrap the
text there is n

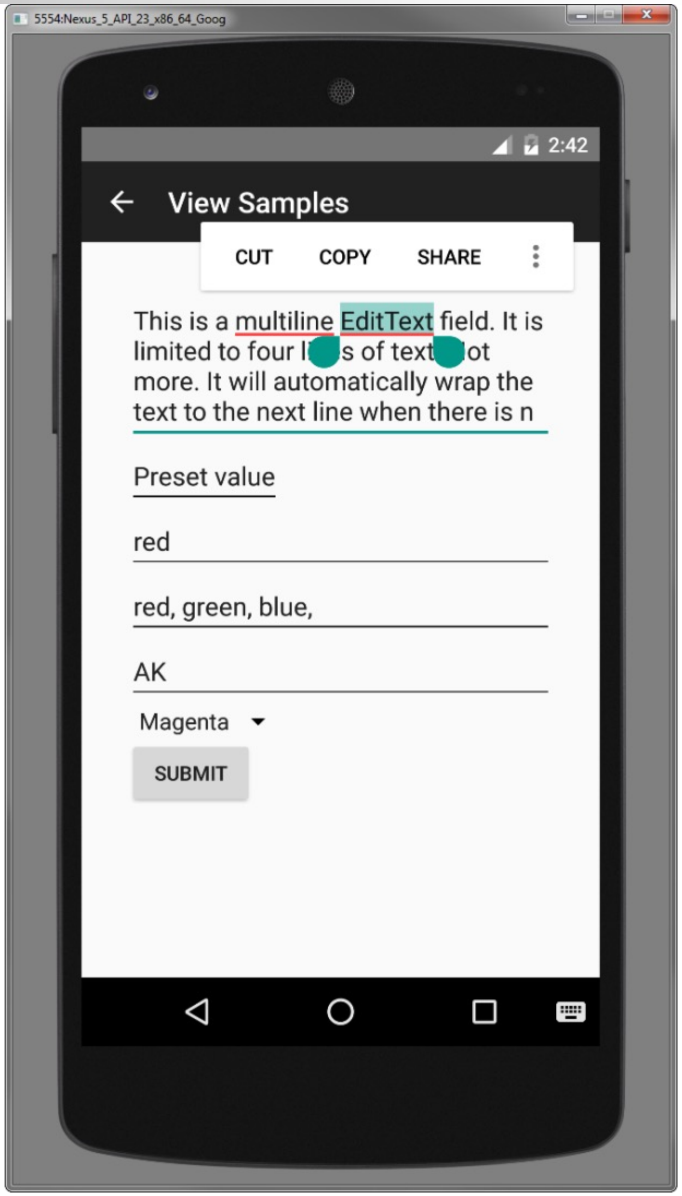
Pres

red

red,

AK

Ma



This is a multiline EditText field. It is
limited to four l s of text ot
more. It will automatically wrap the
text to the next line when there is n

Preset value

red

red, green, blue,

AK

Magenta

EditText InputTypes

- EditText has a few specific subtypes, each for a different type of input
- They are specified with the `android:inputType` attribute
  - number, numberSigned, numberDecimal
  - phone
  - textPassword
  - textEmailAddress
  - datetime
  - many others

Constraining User Input with Input Filters

- There are times when you don't want the user to type just any string.
  - Validating input after the user has entered something is one way to do this.
  - A better way to avoid wasting the user's time is to filter the input.
  - The `EditText` control provides a way to set an `InputFilter` that does this.

Constraining User Input with Input Filters

- The Android SDK provides `InputFilter` objects.
  - `InputFilter` objects enforce such rules as allowing only uppercase text or limiting the length of the text entered
  - You can create custom filters by implementing the `InputFilter` interface, which contains the single method called `filter()`.

Constraining User Input with Input Filters

```
final EditText text_filtered =
        findViewById(R.id.input_filtered);
text_filtered.setFilters( new
      InputFilter[] {
           new InputFilter.AllCaps(),
           new InputFilter.LengthFilter(2)
} );
```

Helping the User with Autocompletion

- The Android SDK also provides a way to help the user with entering commonly used data into forms.
  - This functionality is provided through the autocomplete feature.

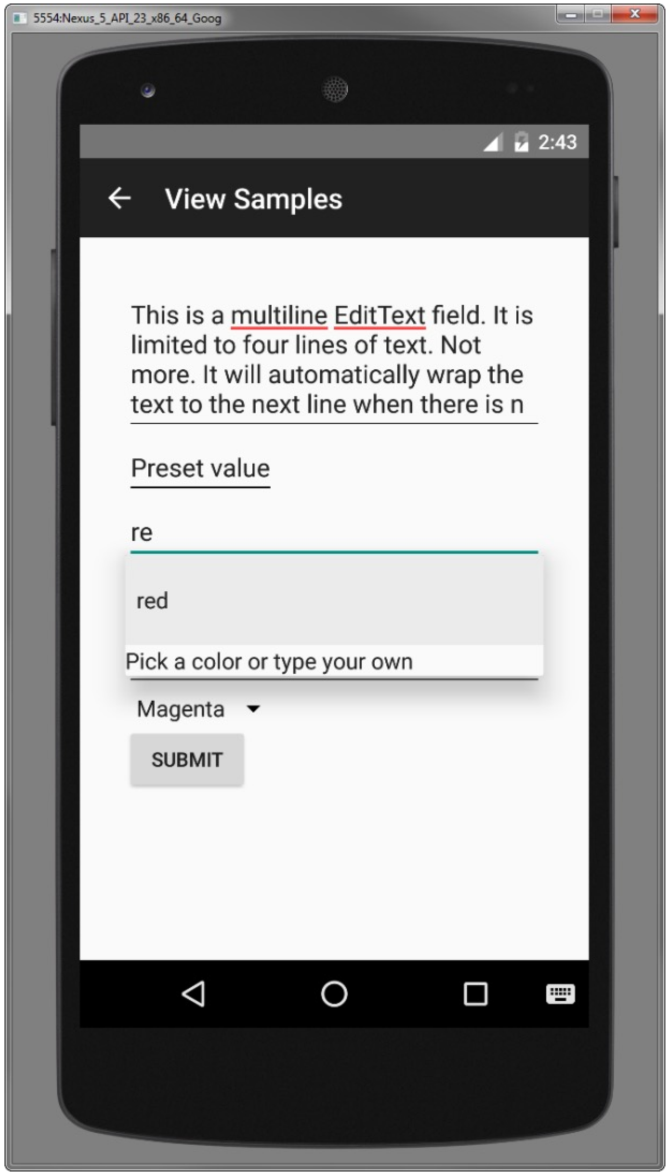
Helping the User with Autocompletion

- There are two forms of autocomplete:
  - One is the more standard style of filling in the entire text entry based on what the user types.
  - If the user begins typing a string that matches a word in a developer-provided list, the user can choose to complete the word with a tap.
  - This is done through the AutoCompleteTextView control.

Helping the User with Autocompletion

- The second method allows the user to enter a list of items, each of which has autocomplete functionality.
  - These items must be separated in some way by providing a `Tokenizer` to the `MultiAutoCompleteTextView` object.
  - A common `Tokenizer` implementation is provided for comma-separated lists using the `MultiAutoCompleteTextView.CommaTokenizer` object.

Helping the User with Autocompletion


5554:Nexus_5_API_23_x86_64_Goog

2:43

View Samples

This is a multiline EditText field. It is limited to four lines of text. Not more. It will automatically wrap the text to the next line when there is n

Preset value

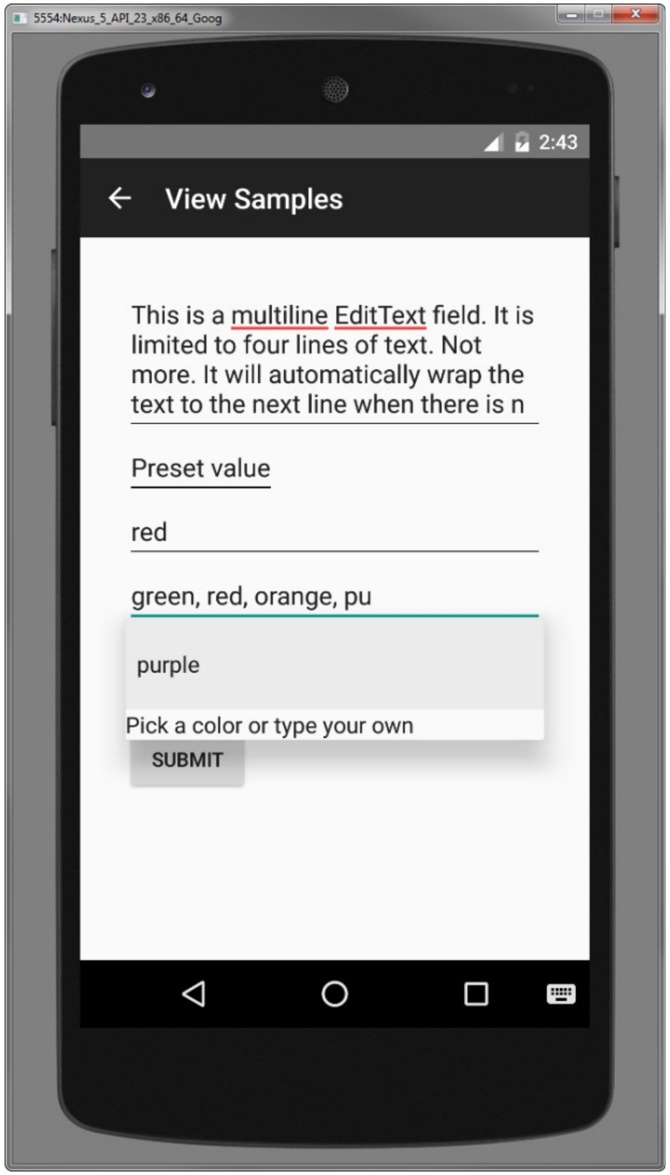
re

red

Pick a color or type your own

Magenta

SUBMIT


5554:Nexus_5_API_23_x86_64_Goog

2:43

View Samples

This is a multiline EditText field. It is limited to four lines of text. Not more. It will automatically wrap the text to the next line when there is n

Preset value

red

green, red, orange, pu

purple

Pick a color or type your own

SUBMIT

Helping the User with Autocompletion

- Both of the autocomplete text editors use an Adapter to get the list of text they use to provide completions to the user.

Helping the User with Autocompletion

- This example shows how to provide an `AutoCompleteTextView` that can help users type some of the basic colors from an array in the code:

```
final String[] COLORS = {
    "red", "green", "orange", "blue", "purple",
    "black", "yellow", "cyan", "magenta" };
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line,
        COLORS);
AutoCompleteTextView text = (AutoCompleteTextView)
    findViewById(R.id.AutoCompleteTextView01);
text.setAdapter(adapter);
```

Helping the User with Autocompletion

```
<AutoCompleteTextView
    android:id="@+id/AutoCompleteTextView01"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:completionHint="Pick a color or type your own"
    android:completionThreshold="1" />
```

Helping the User with Autocompletion

- The `MultiAutoCompleteTextView` is essentially the same as the regular autocomplete, except that you must assign a Tokenizer to it so that the control knows where each autocompletion should begin.

Helping the User with Autocompletion

- The following is an example that uses the same Adapter as the previous example but includes a Tokenizer for a list of user color responses, each separated by a comma:

```
MultiAutoCompleteTextView mtext =
        (MultiAutoCompleteTextView)
                findViewById(R.id.MultiAutoCompleteTextView01);
mtext.setAdapter(adapter);
mtext.setTokenizer(new
                MultiAutoCompleteTextView.CommaTokenizer());
```

Displaying Text to Users with TextView

- One of the most basic user interface elements, or controls, in the Android SDK is the `TextView` control.
  - You primarily use it to display fixed text strings or labels.
- The `TextView` control is a child control within other screen elements and controls.
- As with most of the user interface elements, it is derived from View and is within the

  `android.widget` package.
- Because it is a `View`, all the standard attributes such as width, height, padding, and visibility can be applied to the object.

Displaying Text to Users with TextView

- However, because this is a text-displaying control, you can apply many other `TextView` attributes to control behavior and how the text is viewed in a variety of situations.
- `<TextView>` is the XML layout file tag used to display text on the screen.
- You can set the `android:text` property of the `TextView` to be either a raw text string in the layout file or a reference to a string resource.

Displaying Text to Users with TextView

```
<TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Some text for display here" />

<TextView
    android:id="@+id/TextView02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sample_text" />
```

The last example references one of the string resources (sample_text), which we will discuss later.

Displaying Text to Users with TextView

- You can change the text displayed programmatically by calling the `setText()` method on the TextView object.
- Retrieving the text is done with the `getText()` method.

Configuring Layout and Sizing

- The `TextView` control has a number of attributes that dictate how the text is drawn and flows.
- You can set the `TextView` to be just a single line and a fixed width.
- If you enter a long string of text that can’t fit, the text truncates abruptly.
- Special attributes can handle this problem.

Configuring Layout and Sizing

- The width of a TextView can be specified in terms of ems rather than in pixels.
- An em is a term used in typography that is defined in terms of the point size of a particular font.
  - For example, the measure of an em in a 12-point font is 12 points (in digital typography, 1 pt is 1/72 of an inch).
  - Through the ems attribute, you can set the desired width of a `TextView`.
  - Additionally, you can use the `maxEms` and `minEms` attributes to set the maximum width and minimum width, respectively, of the `TextView` in terms of ems.

Configuring Layout and Sizing

- The height of a `TextView` can be set in terms of lines of text rather than pixels.
  - This is useful for controlling how much text can be viewed regardless of the font size.
  - The lines attribute sets the number of lines that the `TextView` can display.
  - You can also use `maxLines` and `minLines` to control the maximum height and minimum height, respectively, that the `TextView` displays.

Configuring Layout and Sizing

```
<TextView
    android:id="@+id/TextView04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:lines="2"
    android:ems="12"
    android:text="@string/autolink_test" />
```

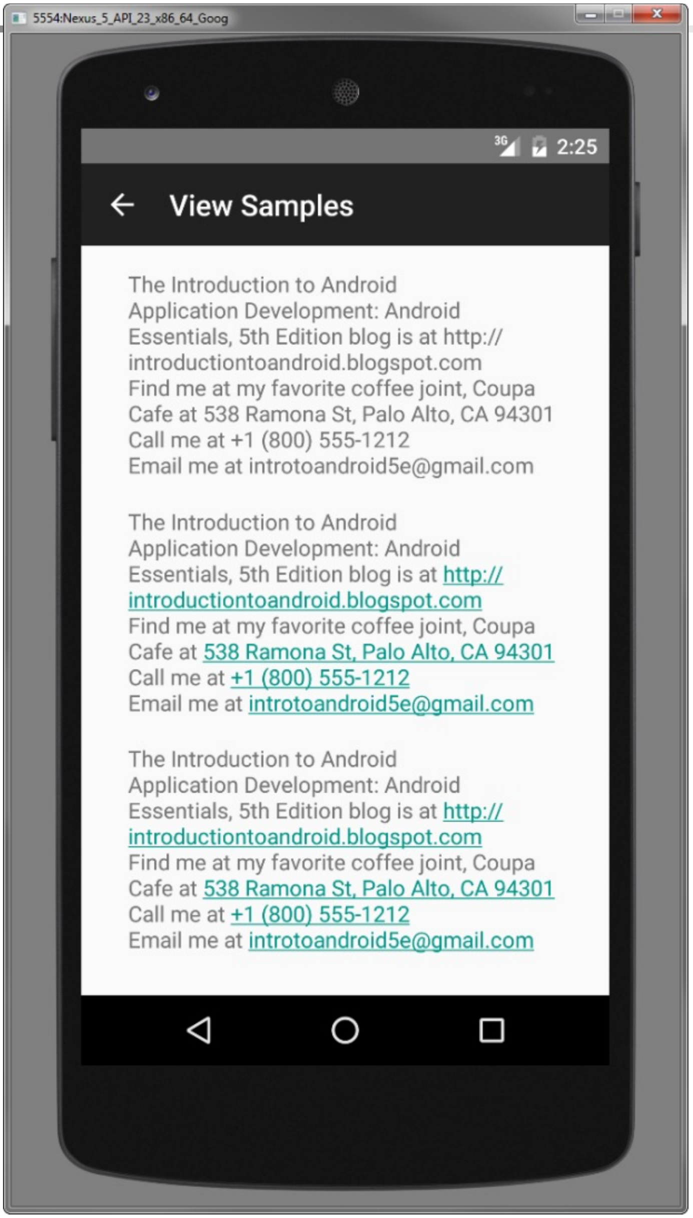
Creating Contextual Links in Text

- If your text contains references to email addresses, Web pages, phone numbers, or even street addresses, you might want to consider using the attribute `autoLink`.
- The `autoLink` attribute has four values that you can use in combination with each other.
- When enabled, these `autoLink` attribute values create standard Web-style links to the application that can act on that data type.
- For instance, setting the attribute to web automatically finds and links any URLs to Web pages.

Creating Contextual Links in Text

- Your text can contain the following values for the `autoLink` attribute:
  - `none`: disables all linking
  - `web`: enables linking of URLs to Web pages
  - `email`: enables linking of email addresses to the mail client with the recipient filled in
  - `phone`: enables linking of phone numbers to the dialer application with the phone number filled in, ready to be dialed
  - `map`: enables linking of street addresses to the map application to show the location
  - `all`: enables all types of linking

Creating Contextual Links in Text



The Introduction to Android Application Development: Android Essentials, 5th Edition blog is at http://introductiontoandroid.blogspot.com
Find me at my favorite coffee joint, Coupa Cafe at 538 Ramona St, Palo Alto, CA 94301
Call me at +1 (800) 555-1212
Email me at introtoandroid5e@gmail.com

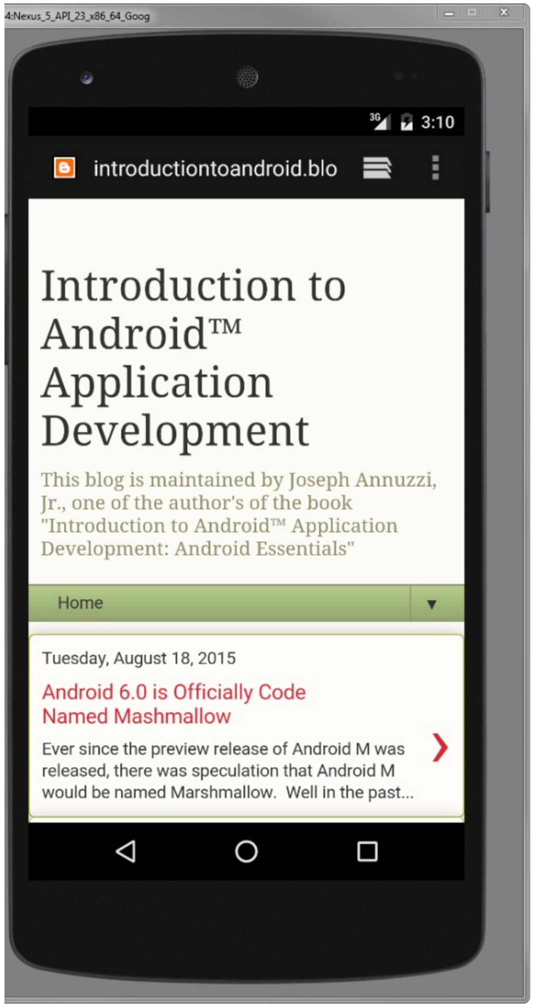
The Introduction to Android Application Development: Android Essentials, 5th Edition blog is at http://introductiontoandroid.blogspot.com
Find me at my favorite coffee joint, Coupa Cafe at 538 Ramona St, Palo Alto, CA 94301
Call me at +1 (800) 555-1212
Email me at introtoandroid5e@gmail.com

The Introduction to Android Application Development: Android Essentials, 5th Edition blog is at http://introductiontoandroid.blogspot.com
Find me at my favorite coffee joint, Coupa Cafe at 538 Ramona St, Palo Alto, CA 94301
Call me at +1 (800) 555-1212
Email me at introtoandroid5e@gmail.com

Creating Contextual Links in Text

```
<TextView
    android:id="@+id/TextView02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/autolink_test"
    android:autoLink="web|email" />
```

Creating Contextual Links in Text


4:Nexus_5_API_23_x86_64_Goog

3:10

introductiontoandroid.blo

Introduction to Android™ Application Development

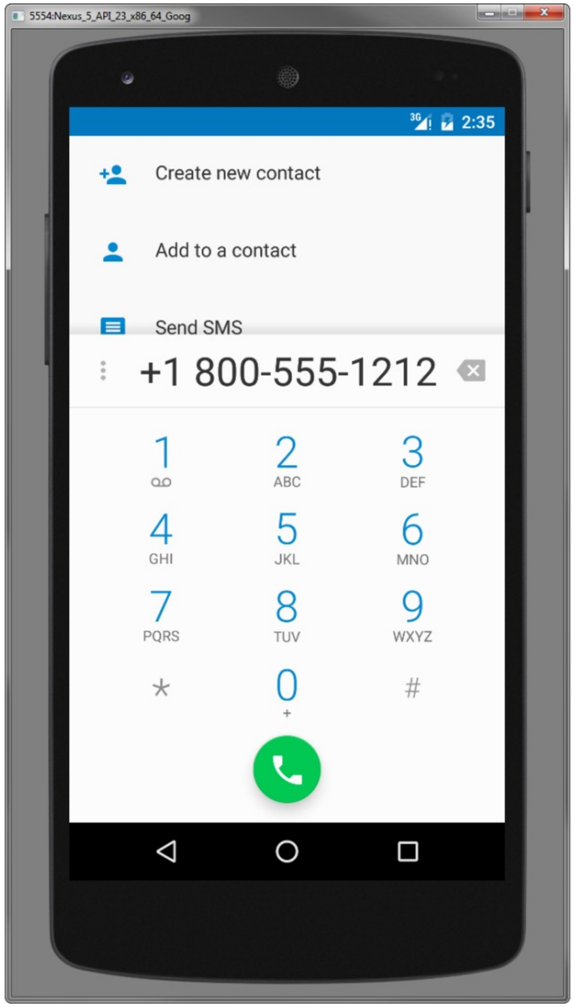
This blog is maintained by Joseph Annuzzi, Jr., one of the author's of the book "Introduction to Android™ Application Development: Android Essentials"

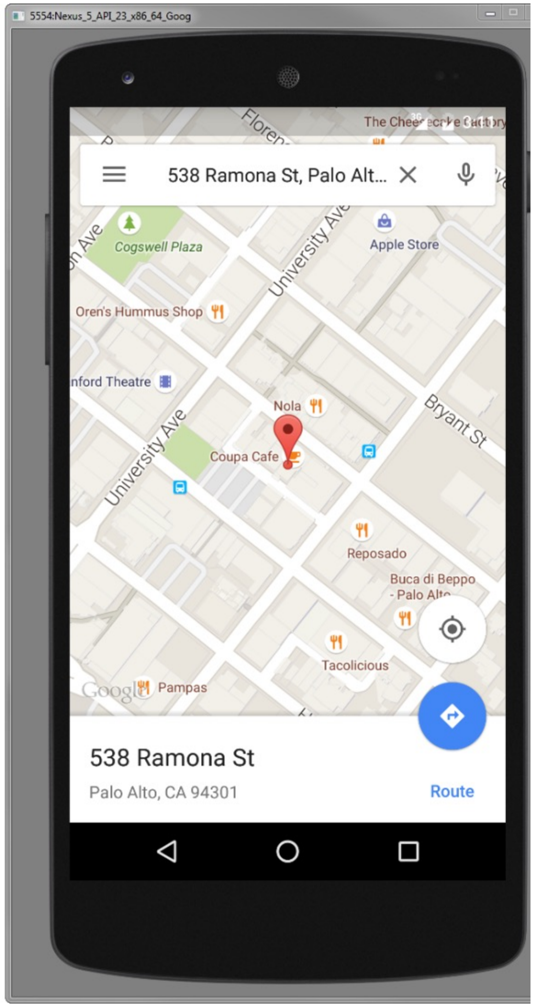
Home ▾

Tuesday, August 18, 2015

Android 6.0 is Officially Code Named Mashmallow

Ever since the preview release of Android M was released, there was speculation that Android M would be named Marshmallow. Well in the past...





Giving Users Choices Using Spinner Controls

- Sometimes you want to limit the choices available for users to type.
  - For instance, if users are going to enter the name of a state, you might as well limit them to only the valid states, because this is a known set.
  - Although you could do this by letting them type something and then blocking invalid entries, you can also provide similar functionality with a `Spinner` control.

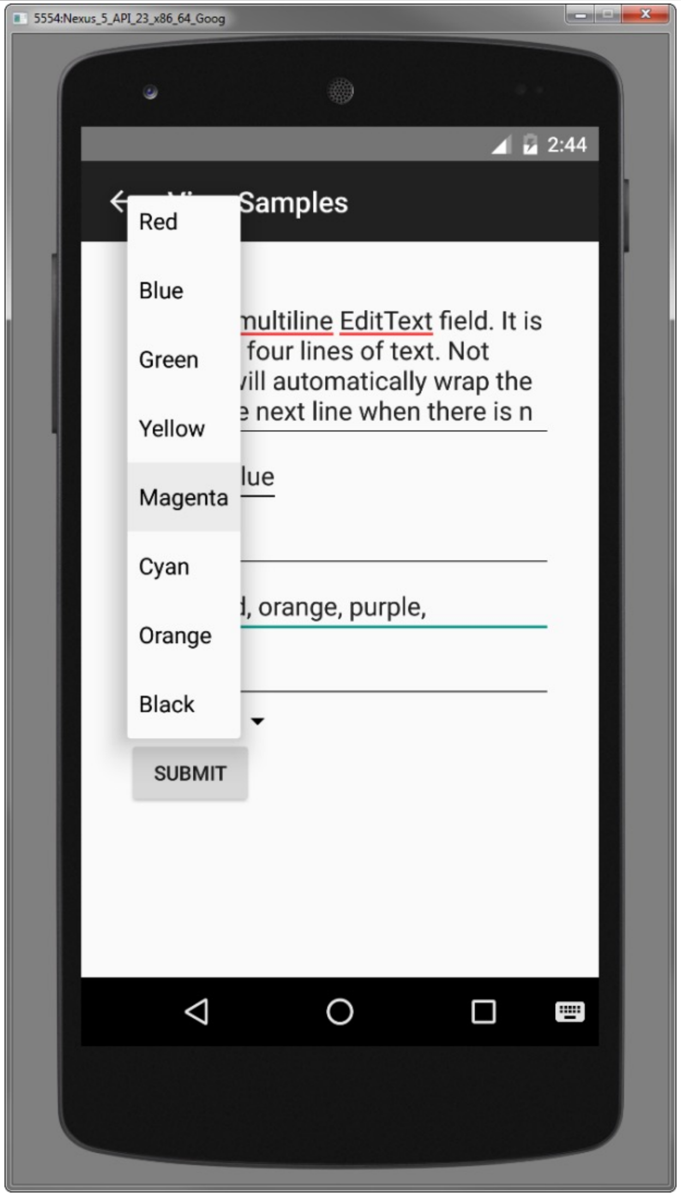
Giving Users Choices Using Spinner Controls

- As with the autocomplete method, the possible choices for a `Spinner` can come from an Adapter.
- It is also possible to set the available choices in the layout definition by using the entries attribute with an array resource.
  - Specifically, this is a string array that is referenced as something, such as `@array/state-list`.
- The `Spinner` control isn't actually an `EditText`, although it is frequently used in a similar fashion.

Giving Users Choices Using Spinner Controls

```
<Spinner
    android:id="@+id/Spinner01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/colors"
    android:prompt="@string/spin_prompt" />
```

Giving Users Choices Using Spinner Controls



multiline EditText field. It is four lines of text. Not will automatically wrap the e next line when there is n

lue

l, orange, purple,

Giving Users Choices Using Spinner Controls

- Because the Spinner control is not a `TextView` but a list of `TextView` objects, it is not possible to directly request the selected text from it.
- Instead, the app must retrieve the specific selected option (each of which is a `TextView` control) and extract the text directly from it

Giving Users Choices Using Spinner Controls

- For example:

```
Spinner spin = findViewById(R.id.Spinner1);
TextView text_sel = spin.getSelectedView();
String selected_text = text_sel.getText().toString();
```

- Alternatively, we could call the `getSelectedItem()` or `getSelectedItemId()` method to deal with other forms of selection.

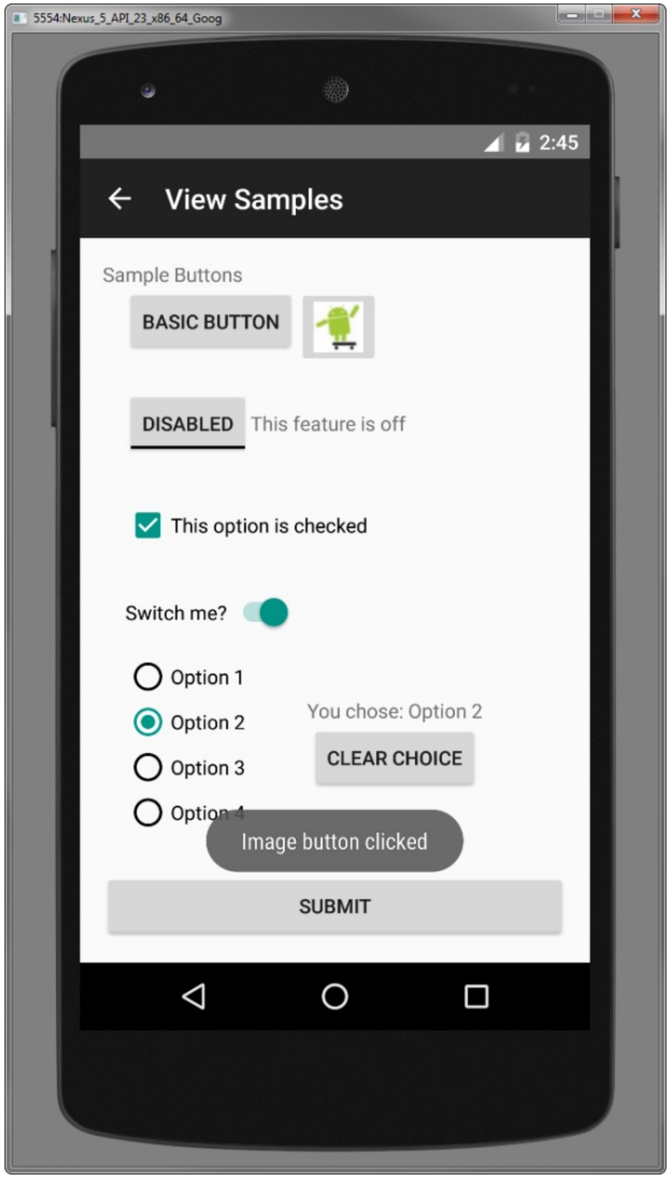
Selections: Buttons, Check Boxes, Switches, Radio Groups

- Other common UI elements are the basic `Button`, `CheckBox`, `ToggleButton`, and `RadioButton`.
- A basic `Button` is often used to perform some sort of action, such as submitting a form or confirming a selection.
- A `CheckBox` is a button with two states—checked and unchecked.

Selections: Buttons, Check Boxes, Switches, Radio Groups

- A `ToggleButton` is similar to a `CheckBox`, but you use it to show the state visually.
- A `Switch` is similar to a `CheckBox`, in that it is a two-state control.
- A `Switch` should be used to present a single choice, while `CheckBox`es are usually shown as a group.
- A `RadioButton` provides an exclusive selection of an item out of a group of items.

Selections: Buttons, Check Boxes, Switches, Radio Groups



Using Basic Buttons

- The `android.widget.Button` class provides a basic Button implementation in the Android SDK.
- Within the XML layout resources, buttons are specified using the `Button` element.
- The primary attribute for a basic `Button` is the text field.
- This is the label that appears on the middle of the button's face.
- You often use basic `Button` controls for buttons with text such as "OK," "Cancel," or "Submit."

Using Basic Buttons

```
<Button
    android:id="@+id/basic_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Basic Button" />
```

Using Basic Buttons

```
setContentView( R.layout.buttons );
final Button basic_button = (Button)
                                        findViewById(R.id.basic_button);
basic_button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Toast.makeText( ButtonsActivity.this,
                                "Button clicked”,
                                Toast.LENGTH_SHORT ).show();
    }
});
```

- A `Toast` is a small pop-up message.

Using Basic Buttons

- A Button-like control whose primary label is an image is an `ImageButton`.
- An `ImageButton` is almost exactly like a basic Button.
- Click actions are handled in the same way.
- The primary difference is that you can set its `src` attribute to be an image.

Using Basic Buttons

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/image_button"
    android:src="@drawable/droid"
    android:contentDescription="@string/droidSkater"/>
```

Using CheckBox and ToggleButton Controls

- The `CheckBox` button is often used in lists of items where the user can select multiple items.
- The Android `CheckBox` contains a text attribute that appears to the side of the check box.
- Because the `CheckBox` class is derived from the `TextView` and `Button` classes, many of the attributes and methods behave in a similar fashion.

Using CheckBox and ToggleButton Controls

```
<CheckBox
    android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Check me?" />
```

Using CheckBox and ToggleButton Controls

```
CheckBox check_button = findViewById(R.id.checkbox);

check_button.setOnClickListener(new View.OnClickListener() {
    public void onClick (View v) {
        CheckBox cb = findViewById(R.id.checkbox);
        cb.setText( check_button.isChecked() ?
                        "This option is checked" :
                        "This option is not checked");
    }
});
```

Using CheckBox and ToggleButton Controls

- A ToggleButton is similar to a `CheckBox` in behavior but is usually used to show or alter the “on” or “off” state of something.
- Like the `CheckBox`, it has a state (checked or not).
- Also like the `CheckBox`, the act of changing what displays on the button is handled for us.

Using CheckBox and ToggleButton Controls

- Unlike the `CheckBox`, it does not show text next to it. Instead, it has two text fields:
  - The first attribute is `textOn`, which is the text that displays on the button when its checked state is on.
  - The second attribute is `textOff`, which is the text that displays on the button when its checked state is off.
  - The default text for these is "ON" and "OFF," respectively.

Using CheckBox and ToggleButton Controls

```
<ToggleButton
    android:id="@+id/toggle_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Toggle"
    android:textOff="Disabled"
    android:textOn="Enabled" />
```

Using CheckBox and ToggleButton Controls

- The Switch control, `android.widget.Switch`, provides similar two-state behavior to the `ToggleButton` control, only instead of the control being clicked to toggle between the states, it looks more like a slider.
  - The `Switch` control was introduced in API Level 14.

Using CheckBox and ToggleButton Controls

```
<Switch android:id="@+id/switch1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Switch me?"
    android:textOn="Wax On"
    android:textOff="Wax Off" />
```

Using RadioGroup and RadioButton

- You often use radio buttons when a user should be allowed to select only one item from a small group of items.
  - For instance, a question asking for gender can give three options: male, female, and unspecified.
  - Only one of these options should be checked at a time.

Using RadioGroup and RadioButton

- The `RadioButton` objects are similar to `CheckBox` objects.
  - They have a text label next to them, set via the text attribute, and they have a state (checked or unchecked).
  - However, you can group `RadioButton` objects inside a `RadioGroup` that handles enforcing their combined states so that only one `RadioButton` can be checked at a time.
  - If the user selects a `RadioButton` that is already checked, it does not become unchecked.
  - You can provide the user with an action to clear the state of the entire `RadioGroup` so that none of the buttons are checked.

Using RadioGroup and RadioButton

```
<RadioGroup
   android:id="@+id/RadioGroup01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
   <RadioButton android:id="@+id/RadioButton01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Option 1" />
   <RadioButton android:id="@+id/RadioButton02"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Option 2" />
   <RadioButton android:id="@+id/RadioButton03"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Option 3" />
   <RadioButton android:id="@+id/RadioButton04"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Option 4" />
</RadioGroup>
```

Using RadioGroup and RadioButton

```
final RadioGroup group = findViewById(R.id.RadioGroup01);
final TextView tv = findViewById(R.id.TextView01);

group.setOnCheckedChangeListener(new
    RadioGroup.OnCheckedChangeListener() {
        public void onCheckedChanged(
            RadioGroup group, int checkedId) {
            if (checkedId != -1) {
                RadioButton rb = findViewById(checkedId);
                if (rb != null) {
                    tv.setText("You chose: " + rb.getText());
                }
            } else {
                tv.setText("Choose 1");
            }
        }
});
```

Using RadioGroup and RadioButton

- Clearing the selection

```
Button clear_choice = findViewById(R.id.Button01);
clear_choice.setOnClickListener( new View.OnClickListener() {
    public void onClick(View v) {
        RadioGroup group = findViewById(R.id.RadioGroup01);
        if (group != null) {
            group.clearCheck();
        }
    }
}
```

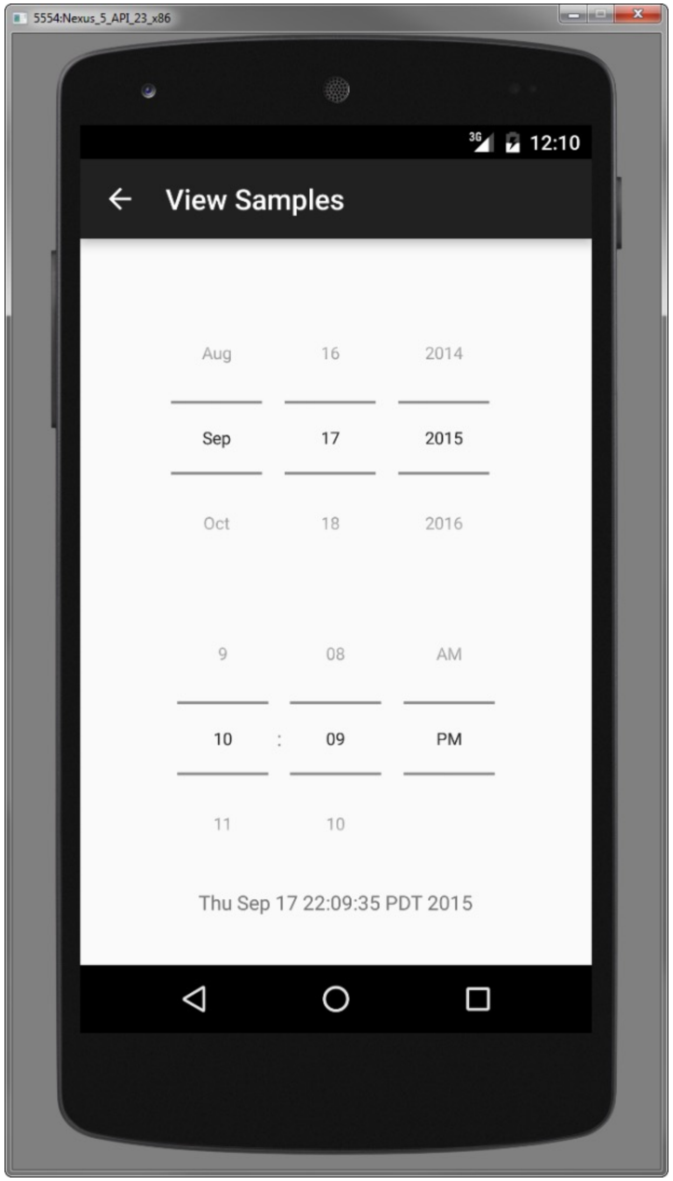
Using ImageView

```
<ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="40dp"
        android:paddingRight="40dp"
        android:scaleType="fitCenter"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/uga_seal" />
```

Retrieving Dates and Times from Users

- The Android SDK provides a couple of controls for getting date and time input from the user.
- One particular control is the `DatePicker` control.
  - It can be used to get a month, day, and year from the user.

Retrieving Dates and Times from Users



Thu Sep 17 22:09:35 PDT 2015

Retrieving Dates and Times from Users

```
<DatePicker
    android:id="@+id/DatePicker01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false‟
    android:datePickerMode=”spinner”
    android:spinnersShown="true" />
```

or

```
    android:datePickerMode=”calendar”
```

and/or

```
    android:calendarViewShown="false” or ”false”
```

Retrieving Dates and Times from Users

```
final DatePicker date = findViewById(R.id.DatePicker01);
date.init( 2015, 7, 17,
    new DatePicker.OnDateChangedListener() {
        public void onDateChanged( DatePicker view, int year,
            int monthOfYear, int dayOfMonth ) {
            Calendar calendar = Calendar.getInstance();
            calendar.set( year,
                    monthOfYear,
                    dayOfMonth,
                    time.getCurrentHour(),
                    time.getCurrentMinute() );
            text.setText( calendar.getTime().toString() );
        }
});
```

Retrieving Dates and Times from Users

```
time.setOnTimeChangedListener(new
TimePicker.OnTimeChangedListener() {
    public void onTimeChanged( TimePicker view,
        int hourOfDay, int minute ) {
        Calendar calendar = Calendar.getInstance();
        calendar.set( calendar.get( Calendar.YEAR ),
                calendar.get( Calendar.MONTH ),
                calendar.get( Calendar.DAY_OF_MONTH ),
                hourOfDay,
                minute );
        text.setText( calendar.getTime().toString() );
    }
});
```

Retrieving Dates and Times from Users

- Android also provides a `NumberPicker` widget, which is very similar to the `TimePicker` widget.
- You can use a `NumberPicker` to present to users a selection mechanism for choosing a number from a predefined range.
- There are two different types of `NumberPicker` you can present; both are entirely based on the theme your application is using.
- To learn more about the `NumberPicker`, see:
  - http://d.android.com/reference/android/widget/NumberPicker.html

Using Indicators to Display Progress and Activity to Users

- The Android SDK provides a number of controls that can be used to show some form of activity-in-progress information to the user.
- These indicator controls include the `ProgressBar`, clocks, and other similar controls.

Indicating Progress with ProgressBar

- Applications commonly perform actions that can take a while.
- A good practice during this time is to show users some sort of progress indicator that informs them that the application is off "doing something."
- Applications can also show how far a user has progressed through some operation.

Indicating Progress with ProgressBar

- The Android SDK provides several types of `ProgressBar`.
  - The standard `ProgressBar` is a circular indicator that only animates.
    - It does not show how complete an action is.
    - It can, however, show that something is taking place.
      - This is useful when an action is indeterminate in length.
    - There are three sizes for this type of progress indicator.

Indicating Progress with ProgressBar

- The second type is a horizontal `ProgressBar` that shows the completeness of an action.
  - For example, you can see how much of a file has downloaded.
  - The horizontal `ProgressBar` can also have a secondary progress indicator on it.
  - This can be used to show the completion of a downloading media file while that file plays.

Indicating Progress with ProgressBar

```
<ProgressBar
    android:id="@+id/progress_bar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Indicating Progress with ProgressBar

- The default style is for a medium-size circular progress indicator.
  - This is not a “bar” at all.
- The other two styles for indeterminate `ProgressBar` are `progressBarStyleLarge` and `progressBarStyleSmall`.
- These styles animate automatically.

Indicating Progress with ProgressBar

```
<ProgressBar
    android:id="@+id/progress_bar"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width=" match_parent"
    android:layout_height="wrap_content"
    android:max="100" />
```

Indicating Progress with ProgressBar

- We can set the indicator progress status programmatically as follows:

```
mProgress = findViewById(R.id.progress_bar);
mProgress.setProgress(75);
```

Adding Progress Indicators to the ActionBar

- You can also put a `ProgressBar` in your application’s title bar (on top of the screen).
- This can save screen real estate and can also make it easy to turn an indeterminate progress indicator on and off without changing the look of the screen.
- Indeterminate progress indicators are commonly used to display progress on pages where items need to be loaded before the page can finish drawing.
- This is often employed on Web browser screens.

Adding Progress Indicators to the ActionBar

```
<android.support.v7.widget.Toolbar xmlns:app="http://schemas.android.com/apk/res-
auto"
    android:id="@+id/toolbar_progress"
    android:background="@color/bg_color"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?attr/actionBarSize"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
    app:theme="@style/ToolbarTheme">
    <ProgressBar
        android:id="@+id/toolbar_spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end"
        android:indeterminate="true"
        android:visibility="gone" />
</android.support.v7.widget.Toolbar>
```

Adding Progress Indicators to the ActionBar

```
supportRequestWindowFeature(Window.
                FEATURE_INDETERMINATE_PROGRESS);
supportRequestWindowFeature(Window.FEATURE_PROGRESS);
setContentView(R.layout.indicators);
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar_progress);
toolbar.setTitleTextColor(Color.WHITE);
setSupportActionBar(toolbar);
if (getSupportActionBar() != null) {
  getSupportActionBar().setDisplayHomeAsUpEnabled(true);
}
ProgressBar toolbarProgress = findViewById(R.id.toolbar_spinner);
toolbarProgress.setVisibility(View.VISIBLE);
toolbarProgress.setProgress(5000);
```

Adding Progress Indicators to the ActionBar

- To use the indeterminate indicator on your Activity object’s `ActionBar`, you need to request the feature `Window.FEATURE_INDETERMINATE_PROGRESS`, as previously shown.
  - This shows a small circular indicator in the right side of the `ActionBar`.
  - For a horizontal `ProgressBar` style that shows behind the `ActionBar`, you need to enable `Window.FEATURE_PROGRESS`.
  - These features must be enabled before your application calls the `setContentView()` method, as shown in the preceding example.

Indicating Activity with Activity Bars and Activity Circles

- When there is no telling how long an operation will take to complete, but you need a way to indicate to the user that an operation is taking place, you should use an activity bar or an activity circle.

Indicating Activity with Activity Bars and Activity Circles

- You define an activity bar or circle exactly like you define a `ProgressBar`, with one small change: you need to tell Android that the operation running will continue for an indeterminate amount of time by either setting the attribute within your layout file using `android:indeterminate`, or from within your code by setting the `ProgressBar`'s visibility to indeterminate using the `setProgressBarIndeterminateVisibility()` method.

Adjusting Progress with SeekBars

- You have seen how to display progress to the user.
- If it is necessary to allow user to adjust the current cursor position in a playing media file or to adjust a volume setting, use a `SeekBar`.
- `SeekBar` control is provided by the Android SDK.

Adjusting Progress with SeekBars

- It’s like the regular horizontal `ProgressBar` but includes a thumb, or selector, that can be dragged by the user.
- A default thumb selector is provided, but you can use any drawable item as a thumb.

Adjusting Progress with SeekBar

```
<SeekBar
    android:id="@+id/seekbar1"
    android:layout_height="wrap_content"
    android:layout_width="240dp"
    android:max="500"
    android:thumb="@drawable/droidsk1" />
```

Adjusting Progress with SeekBar

```
SeekBar seek = findViewById(R.id.seekbar1);
seek.setOnSeekBarChangeListener(
    new SeekBar.OnSeekBarChangeListener() {
        public void onProgressChanged(
            SeekBar seekBar, int progress,boolean
            fromTouch)
        {
            ((TextView)findViewById(R.id.seek_text))
                .setText("Value: "+progress);
            seekBar.setSecondaryProgress(
                (progress+seekBar.getMax())/2);
        }
});
```

Other Valuable User Interface Controls

- Android has a number of other ready-to-use user interface controls to incorporate into your applications.
- The following section is dedicated to introducing the following:
  - RatingBar
  - Time controls, such as
    - Chronometer
    - DigitalClock
    - TextClock
    - AnalogClock

Displaying Rating Data with RatingBar

- Although the `SeekBar` is useful for allowing a user to set a value, such as the volume, the `RatingBar` has a more specific purpose:
  - Showing ratings or getting a rating from a user
- By default, this `ProgressBar` uses the star paradigm, with five stars by default.
- A user can drag across this horizontally to set a rating.
- A program can set the value as well.
- However, the secondary indicator cannot be used because it is used internally by this particular control.

Displaying Rating Data with RatingBar

```
<RatingBar
    android:id="@+id/ratebar1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="4"
    android:stepSize="0.25" />
```

Displaying Rating Data with RatingBar

```
RatingBar rate = findViewById(R.id.ratebar1);
rate.setOnRatingBarChangeListener(new
    RatingBar.OnRatingBarChangeListener() {
    public void onRatingChanged(RatingBar ratingBar,
        float rating, boolean fromTouch) {
        ((TextView)findViewById(R.id.rating_text))
            .setText("Rating: "+ rating);
    }
});
```

Showing Time Passage with the Chronometer

- Sometimes you want to show time passing instead of incremental progress.
- In this case, you can use the `Chronometer` control as a timer.
- This might be useful if it is the user who is taking time doing some task or playing a game where some action needs to be timed.

Showing Time Passage with the Chronometer

```
<Chronometer
    android:id="@+id/Chronometer01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:format="Timer: %s" />
```

Showing Time Passage with the Chronometer

```
final Chronometer timer = findViewById(R.id.Chronometer01);

long base =  timer.getBase();
Log.d(ViewsMenu.debugTag, "base = "+ base);
timer.setBase(0);
timer.start();
```

Displaying the Time

- Displaying the time in an application is often not necessary because Android devices have a status bar to display the current time.
- However, two clock controls are available to display this information:
  - The `TextClock` and `AnalogClock` controls

Using the TextClock

- The `TextClock` control was added in API Level 17.
  - It is meant to be a replacement for the `DigitalClock`, which was deprecated in API Level 17.
- The `TextClock` has many more features than the `DigitalClock`.
  - It allows you to format the display of the date and/or time.
- The `TextClock` allows you to display the time in 12-hour mode or 24-hour mode and even allows you to set the time zone.
- By default, the `TextClock` control does not show the seconds.

Using the TextClock

```
<TextClock
    android:id="@+id/TextClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Using the AnalogClock

- The `AnalogClock` control is a dial-based clock with a basic clock face with two hands.
- It updates automatically as each minute passes.
- The image of the clock scales appropriately with the size of its View.

Using the AnalogClock (Cont'd)

```
<AnalogClock
    android:id="@+id/AnalogClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Playing Video with VideoView

- The `VideoView` control is a video player View used for playing video in your application.
- This View has controls of to play, pause, skip forward, skip backward, and seek.

Playing Video with VideoView

```
<VideoView
    android:id="@+id/video_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Playing Video with VideoView

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_simple_video_view);
    VideoView vv = findViewById(R.id.videoView);
    MediaController mc = new MediaController(this);
    Uri video = Uri.parse("http://andys-veggie-
garden.appspot.com/vid/reveal.mp4");
    vv.setMediaController(mc);
    vv.setVideoURI(video);
}
```

Playing Video with VideoView

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_simple_video_view );
    VideoView vv = findViewById( R.id.videoView );
    MediaController mc = new MediaController( this );
    Uri video = Uri.parse(
     "https://ia802205.us.archive.org/31/items/Unexpect2001/Unexpect2001_512kb.mp4");
    mc.setAnchorView( vv );
    vv.setMediaController( mc );
    vv.setVideoURI( video );
    vv.requestFocus();
    vv.setOnPreparedListener( new MediaPlayer.OnPreparedListener() {
                                             public void onPrepared(MediaPlayer mp) {  vv.start();  }
                                         } );
}
```

Playing Video with VideoView

- You first want to get the `VideoView` from the layout.
  - Then you need to create a `MediaController` object.
- In our case, we are obtaining the video from the Internet.
  - We first need to parse the URL of the video using the `Uri.parse` method so that our code uses a valid Uri object.
- We then use the `setMediaController()` method for adding the `MediaController` object to your `VideoView`, and then we use the `setVideoURI()` method to pass the Uri to our `VideoView`.

Playing Video with VideoView

- In order to play a video from the Internet, add this line to AndroidManifest.xml

```
<uses-permission
    android:name="android.permission.INTERNET" />
```

Summary

- We have learned how to use a `TextView` and how to define many of its attributes.
- We have learned how to use an `EditText` control and how to implement its `InputFilter` interface.
- We have learned about the different types of autocompletion and how to implement an `Adapter` for the different autocomplete views.
- We have learned how to populate `Spinner` controls with different choices and are able to retrieve the selected option.

Summary

- We have learned how to use and implement different forms of user selections, such as buttons, check boxes, switches, and radio groups.
- We are now able to retrieve the date and time from users.
- We are now able to use indicators to display data to users.

References and More Information

- Android API Guides: “User Interface”:
  - http://d.android.com/guide/topics/ui/index.html
- Android SDK Reference regarding the application View class:
  - http://d.android.com/reference/android/view/View.html
- Android SDK Reference regarding the application TextView class:
  - http://d.android.com/reference/android/widget/TextView.html
- Android SDK Reference regarding the application EditText class:
  - http://d.android.com/reference/android/widget/EditText.html

References and More Information

- Android SDK Reference regarding the application Button class:
  - http://d.android.com/reference/android/widget/Button.html
- Android SDK Reference regarding the application CheckBox class:
  - http://d.android.com/reference/android/widget/CheckBox.html
- Android SDK Reference regarding the application Switch class:
  - http://d.android.com/reference/android/widget/Switch.html

References and More Information

- Android SDK Reference regarding the application RadioGroup class:
  - http://d.android.com/reference/android/widget/Switch.html
- Android SDK Reference regarding the support v7 Toolbar class:
  - http://d.android.com/reference/android/support/v7/widget/Toolbar.html
- Android SDK Reference regarding the application VideoView class:
  - http://d.android.com/reference/android/widget/VideoView.html