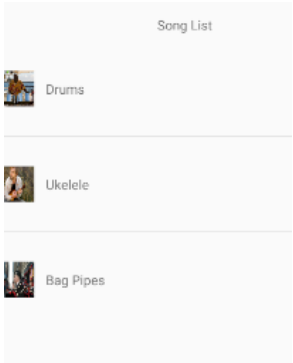
CSIS 3175 ANNOUNCEMENTS

- Midterm: Tuesday, start of class Feb 25, 9:30 am
- Duration: 2 hours
- Content: Lectures 1 – 6 (all demos and notes from class)
- Format: To be done on the computer – only practical, uploaded on blackboard as a compressed file
- Cheat Sheet: 2 sided regular letter size, 2 sheets (4 sides total) – any font, printed/written, whatever you want on it

Lab: Review of ListView

- Create a listview using custom adapter as shown below:

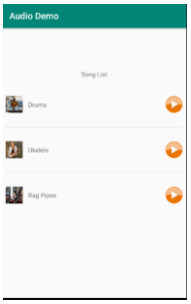


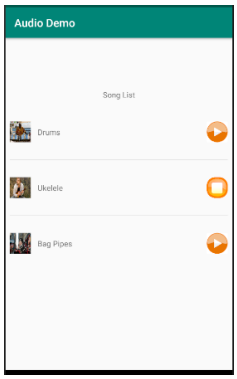
Android Boot Camp for Developers Using Java, 3E

Chapter 6: Jam!
Implementing Audio in Android Apps

We will create a music player App

- ListView with song description, song images, and play/stop images
- Play clicked song if not already playing (stop other song if playing)
- Stop clicked song if it is currently playing





Objectives

In this chapter (slightly different example with media files), you learn to:

- Create an Android project using a splash screen
- Design a TextView control with a background image
- Pause the execution of an Activity with a timer
- Understand the Activity life cycle
- Open an Activity with onCreate( )
- End an Activity with finish( )
- Assign class variables
- Create a raw folder for music files

Objectives

- Play music with a MediaPlayer method
- Start and resume music playback using the start() and pause() methods
- Change the Text property of a control
- Change the visibility of a control

Implementing Audio

- The most common Smartphone activities
  - Texting
  - Talking
  - Gaming
  - Apps
  - Multimedia

Figure 6-1 Aloha Music Android app

Figure 6-2 Music played in the Android app

Implementing Audio

Steps to complete the app:

1. Create a splash screen with a timer.
2. Design a TextView control with a background image.
3. Initialize a TimerTask and a timer.
4. Launch a second Activity.
5. Design a second XML layout.
6. Add music files to the raw folder.
7. Initialize the MediaPlayer class.
8. Play and pause music with a Button control.

Creating a Splash Screen

- A Splash Screen is a window that is displayed for a few seconds before the app opens
- The next screen opens automatically
- The Android initializes its resources and loads necessary files while the splash screen is displayed

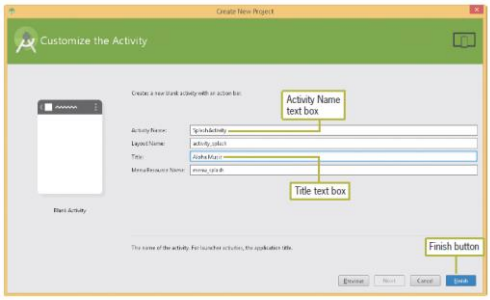


Figure 6-4 Setting up the Splash Activity (splash screen)

Adding a Background Image to a TextView Widget

- Image is <u>not</u> an ImageView control - its a TextView control with a background image

Copies of the three image files appear in the drawable folder (Figure 6-5).

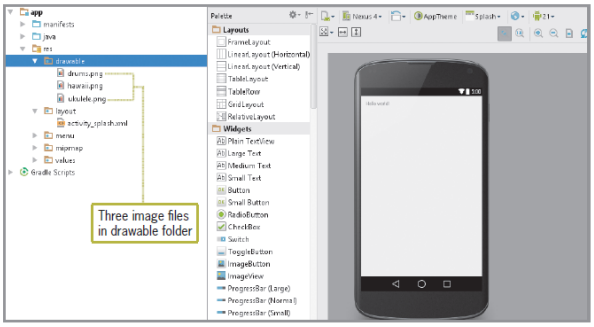


Figure 6-5 Image files in the drawable folder

Adding a Background Image to a TextView Widget

The Translations Editor contains the String values necessary in this app (Figure 6-6).

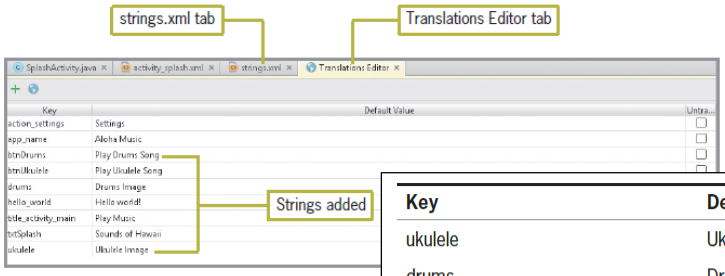


Figure 6-6 Translations Editor

Key	Default Value
ukulele	Ukulele Image
drums	Drums Image
btnUkulele	Play Ukulele Song
btnDrums	Play Drums Song

Table 6-1 Strings for the Aloha Music app

Adding a Background Image to a TextView Widget (continued)

A Plain TextView control with an image background is displayed in the activity_splash.xml file (Figure 6-7).

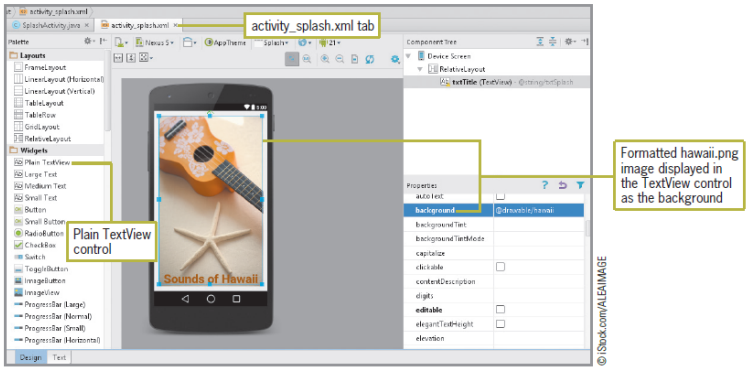


Figure 6-7 activity_splash.xml displays a TextView control with a background image

Creating a Timer

- A **timer** in Java can:
  - Execute a one-time task like displaying an opening splash screen
  - Perform a continuous process such as a morning wake-up call set to run at regular intervals
- Use two Java classes, named **TimerTask** and **Timer**
- Each time a timer runs it runs in a single **thread**
  - A thread is a single sequential flow of control within a program

Creating a Timer (continued)

Code Syntax

```
TimerTask task = new TimerTask( ) {
    @Override
    public void run() {
    // TODO Auto-generated method stub
}
```

The TimerTask class is initiated and a red curly line appears below TimerTask (Figure 6-8).

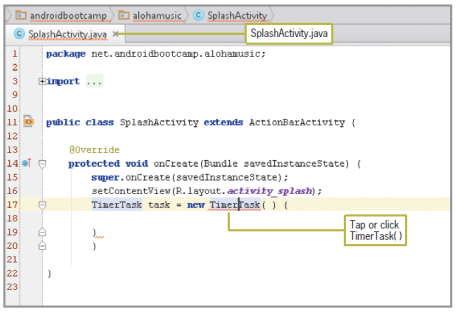


Figure 6-8 setContentView and TimerTask statements

Creating a Timer (continued)

The auto-generated stub for the run( ) method is created automatically for the TimerTask (Figure 6-9).

```
11  public class SplashActivity extends ActionBarActivity {
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_splash);
17          TimerTask task = new TimerTask( ) {
18
19              @Override
20              public void run() {
21
22              }
23          };
24          }
25
26  }
27
```

run( ) method stub

Semicolon ends stub

Figure 6-9 run( ) method

Scheduling a Timer

Code Syntax

```
Timer opening = new Timer();
opening.schedule(task,5000);
```

```
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_splash);
18          TimerTask task = new TimerTask( ) {
19
20              @Override
21              public void run() {
22
23              }
24          };
25          Timer opening = new Timer( );
26          }
27
28  }
29
```

Instance of Timer

Figure 6-10 Timer class

Scheduling a Timer

- Timers are scheduled in milliseconds
- 5000 milliseconds = 5 seconds

Code Syntax

```
Timer opening = new Timer();
opening.schedule(task,5000);
```

```
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_splash);
18          TimerTask task = new TimerTask( ) {
19
20              @Override
21              public void run() {
22
23              }
24          };
25          Timer opening = new Timer( );
26          }
27
28  }
29
```

Instance of Timer

Figure 6-10 Timer class

Scheduling a Timer (continued)

The timer named opening, which lasts five seconds, is scheduled (Figure 6-11).

```
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_splash);
18          TimerTask task = new TimerTask( ) {
19
20              @Override
21              public void run() {
22
23              }
24          };
25          Timer opening = new Timer( );
26          opening.schedule(task,5000);
27          }
28
29  }
30
```

Timer scheduled for 5 seconds

Figure 6-11 Timer scheduled for 5 seconds

Life and Death of an Activity

- Each activity has a **life cycle** – a series of actions from the beginning of an Activity until its end
- When the activity begins, we use an **onCreate() method** to load it into memory
- When the activity ends, we use an **onDestroy() method** to remove it from memory
- Four **states** of an Activity:
  - Active
  - Pause
  - Stopped
  - Dead

Life and Death of an Activity (continued)

Method	Description
onCreate( )	The onCreate( ) method begins each Activity. This method also provides a Bundle containing the Activity's previously frozen state, if it had one.
onRestart( )	If the Activity is stopped, onRestart( ) begins the Activity again. If this method is called, it indicates your Activity is being redisplayed to the user from a stopped state. The onRestart( ) method is always followed by onStart( ).
onStart( )	If the Activity is hidden, onStart( ) makes the Activity visible.
onResume( )	The onResume( ) method is called when the user begins interacting with the Activity. The onResume( ) method is always followed by onPause( ).
onPause( )	This method is called when an Activity is about to resume.
onStop( )	This method hides the Activity.
onDestroy( )	This method destroys the Activity. Typically, the finish( ) method (part of the onDestroy( ) method) is used to declare that the Activity is finished; when the next Activity is called, it releases all the resources from the first Activity.

Table 6-2 Methods used in the life cycle of an Activity

Life and Death of an Activity

- Ovals represent major states of the Activity
- Rectangles represent methods that can be implemented to perform operations

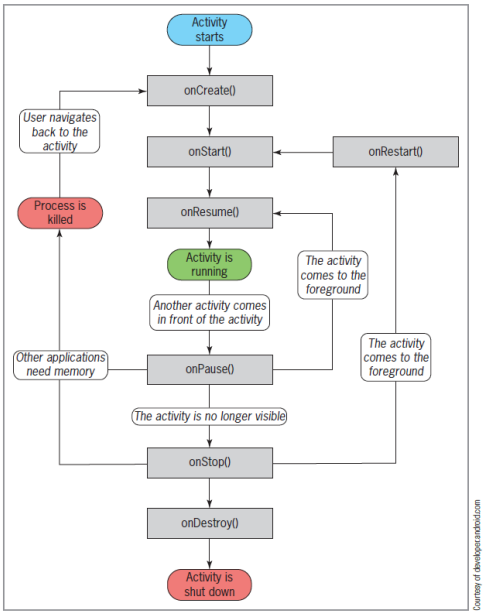


Figure 6-12 Android life cycle

Life and Death of an Activity (continued)

The finish( ) statement releases the resources that were created for the SplashActivity and closes the Activity (Figure 6-13).

```
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_splash);
18          TimerTask task = new TimerTask( ) {
19
20              @Override
21              public void run() {
22                  finish();
23              }
24          };
25          Timer opening = new Timer( );
26          opening.schedule(task,5000);
27          }
28
29      }
30
```

finish( ) method

Figure 6-13 finish( ) method called

Launching the Next Activity

- After the Splash Screen is destroyed an intent must request that the next Activity is launched
- Main.xml already exists as the default layout
- A second class named Main must be created before the code can launch this Java class
- Android manifest file must be updated to include the Main Activity
- Main Activity is responsible for playing music

Launching the Next Activity (continued)

A second class named MainActivity and activity_main.xml are created (Figure 6-14).

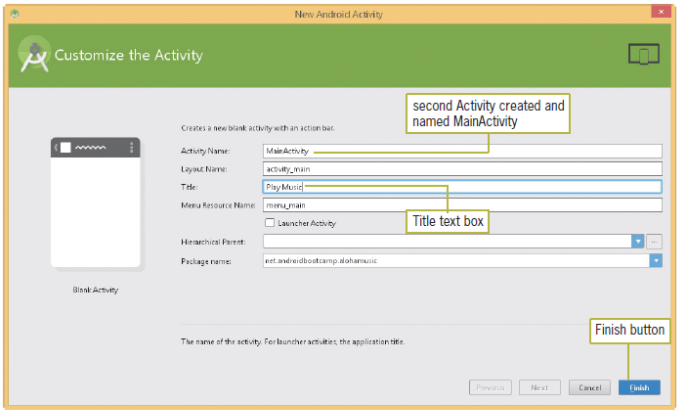


Figure 6-14 Second Activity, which is called MainActivity

Launching the Next Activity (continued)

The second Activity named MainActivity is launched with an Intent statement (Figure 6-15).

Figure 6-15 Launching MainActivity after the Splash screen is displayed for 5 seconds

Designing the activity_main.xml File

The image for the first song named Ukulele is placed in activity_main.xml (Figure 6-16).

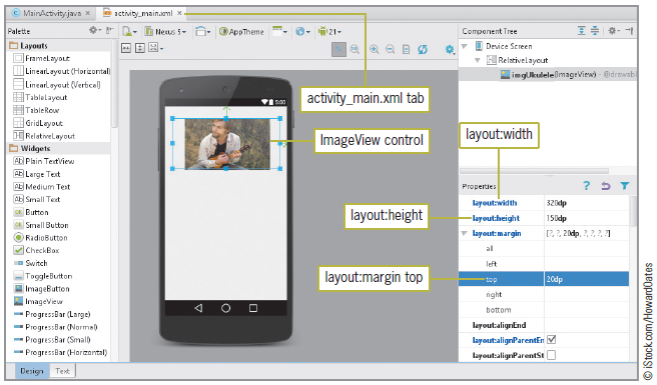


Figure 6-16 Second XML layout

Designing the activity_main.xml File

The button to select the first song named Ukulele is placed on the emulator (Figure 6-17).

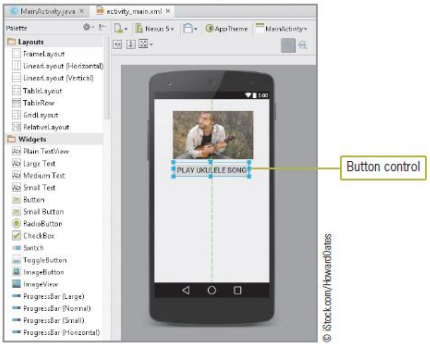


Figure 6-17 Second XML layout, continued

The image and button to select the second song named Drums are designed in activity_main.xml (Figure 6-18).

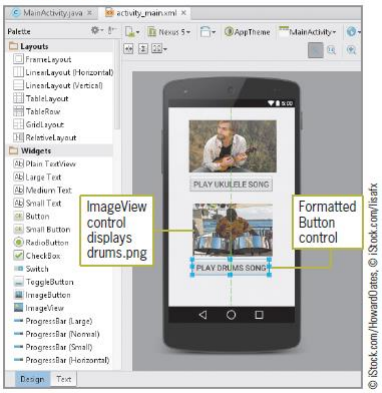


Figure 6-18 activity_main.xml layout complete

Class Variables

- Recall that **local variables** are declared within a method
- The **scope** of a variable refers to the variable's visibility within a class
- When a variable is needed in multiple methods in a class, a global variable is used
- Global variables are called **class variables**

Class Variables

Class variables that can be accessed by the rest of the program are initialized (Figure 6-19).

```
MainActivity.java ×
1   package net.androidbootcamp.alohamusic;
2
3   import ...
10
11
12  public class MainActivity extends ActionBarActivity {
13      Button button1, button2;
14      MediaPlayer mpUkulele, mpDrums;
15      int playing;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21      }
22
23  }
24
```

Button, MediaPlayer, and primitive class variables

Figure 6-19 Class variables

Class Variables (continued)

The Button controls named button1 and button2 are referenced in MainActivity.java (Figure 6-20).

```
MainActivity.java ×
1   package net.androidbootcamp.alohamusic;
2
3   import ...
10
11
12  public class MainActivity extends ActionBarActivity {
13      Button button1, button2;
14      MediaPlayer mpUkulele, mpDrums;
15      int playing;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          button1 = (Button) findViewById(R.id.btnUkulele);
22          button2 = (Button) findViewById(R.id.btnDrums);
23      }
24
25  }
26
```

Button controls instantiated

Figure 6-20 Adding Button controls

Class Variables (continued)

An OnClickListener auto-generated stub appears in the code for the first button (Figure 6-21).

```
MainActivity.java ×
1   package net.androidbootcamp.alohamusic;
2
3   import ...
10
11
12  public class MainActivity extends ActionBarActivity {
13      Button button1, button2;
14      MediaPlayer mpUkulele, mpDrums;
15      int playing;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          button1 = (Button)findViewById(R.id.btnUkulele);
22          button2 = (Button)findViewById(R.id.btnDrums);
23          button1.setOnClickListener(bUkulele);
24
25      }
26      Button.OnClickListener bUkulele= new Button.OnClickListener() {
27      @Override
28      public void onClick(View v) {
29
30      }
31  };
32
33  }
34
```

First Button OnClickListener

Auto-generated stub

Semicolon added

Figure 6-21 Inserting the first Button OnClickListener stub

Class Variables

An OnClickListener auto-generated stub appears in the code for the second button (Figure 6-22)

```
MainActivity.java ×
11
12  public class MainActivity extends ActionBarActivity {
13      Button button1, button2;
14      MediaPlayer mpUkulele, mpDrums;
15      int playing;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          button1 = (Button)findViewById(R.id.btnUkulele);
22          button2 = (Button)findViewById(R.id.btnDrums);
23          button1.setOnClickListener(bUkulele);
24          button2.setOnClickListener(bDrums);
25      }
26      Button.OnClickListener bUkulele= new Button.OnClickListener() {
27      @Override
28      public void onClick(View v) {
29
30          }
31  };
32      Button.OnClickListener bDrums = new Button.OnClickListener( ) {
33
34          @Override
35          public void onClick(View v) {
36
37          }
38      };
39
40  }
41
```

Second Button OnClickListener

Auto-generated stub for the Button OnClickListener

Semicolon added

Figure 6-22 Inserting the second Button OnClickListener stub

Playing Music

- Android phones and tablets have built-in music players
- Androids can play audio and video from several data sources
- .mp3 files are most common
- Can also play .wav, .ogg, and .midi
- Uses **codec** technology to compress and decompress files

Creating a Raw Folder for Music Files

A resource directory named raw is created using the New Resource Directory dialog box (Figure 6-23).

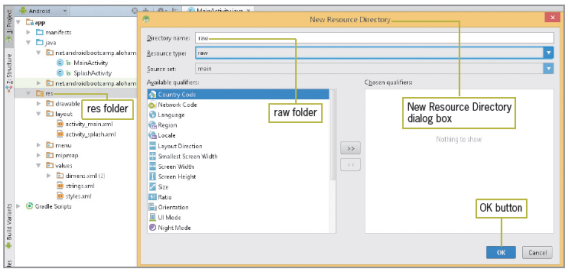


Figure 6-23 The raw folder added for sound files

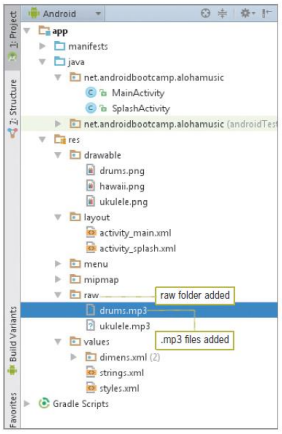


Figure 6-24 MP3 files added to raw folder

Using the MediaPlayer Class

A **MediaPlayer class** provides the methods to control audio playback on Android devices

Code Syntax

```
MediaPlayer mpUkulele = MediaPlayer.create(this, R.raw.ukulele);
```


Using the MediaPlayer Class

The two class variables are assigned an instance of the MediaPlayer class (Figure 6-25).

MainActivity.java

```
12  public class MainActivity extends ActionBarActivity {
13      Button button1, button2;
14      MediaPlayer mpUkulele, mpDrums;
15      int playing;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          button1 = (Button)findViewById(R.id.btnUkulele);
22          button2 = (Button)findViewById(R.id.btnDrums);
23          button1.setOnClickListener(bUkulele);
24          button2.setOnClickListener(bDrums);
25          mpUkulele = new MediaPlayer();
26          mpUkulele = MediaPlayer.create(this, R.raw.ukulele);
27          mpDrums = new MediaPlayer();
28          mpDrums = MediaPlayer.create(this, R.raw.drums);
29      }
30      Button.OnClickListener bUkulele= new Button.OnClickListener() {
31          @Override
32          public void onClick(View v) {
33
34          }
35  });
```

MediaPlayer statements for both songs

Figure 6-25 MediaPlayer instance statements

The MediaPlayer State

```
MainActivity.java ×
12  public class MainActivity extends ActionBarActivity {
13      Button button1, button2;
14      MediaPlayer mpUkulele, mpDrums;
15      int playing;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          button1 = (Button)findViewById(R.id.btnUkulele);
22          button2 = (Button)findViewById(R.id.btnDrums);
23          button1.setOnClickListener(bUkulele);
24          button2.setOnClickListener(bDrums);
25          mpUkulele = new MediaPlayer();
26          mpUkulele = MediaPlayer.create(this, R.raw.ukulele);
27          mpDrums = new MediaPlayer();
28          mpDrums = MediaPlayer.create(this, R.raw.drums);
29          playing = 0;
30      }
31      Button.OnClickListener bUkulele= new Button.OnClickListener() {
32      @Override
33      public void onClick(View v) {
34
35      }
36  };
```

Variable that changes as state of the music changes

Method	Purpose
start( )	Starts media playback
pause( )	Pauses media playback
stop( )	Stops media playback

Table 6-3 Common MediaPlayer states

Figure 6-26 Variable named "playing" is set to 0

The MediaPlayer State

The Switch decision structure that determines the state of the music is coded for the first onClick method (Figure 6-27).

```
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          button1 = (Button)findViewById(R.id.btnUkulele);
22          button2 = (Button)findViewById(R.id.btnDrums);
23          button1.setOnClickListener(bUkulele);
24          button2.setOnClickListener(bDrums);
25          mpUkulele = new MediaPlayer();
26          mpUkulele = MediaPlayer.create(this, R.raw.ukulele);
27          mpDrums = new MediaPlayer();
28          mpDrums = MediaPlayer.create(this, R.raw.drums);
29          playing = 0;
30      }
31      Button.OnClickListener bUkulele= new Button.OnClickListener() {
32      @Override
33      public void onClick(View v) {
34          switch(playing) {
35              case 0:
36                  mpUkulele.start();
37                  playing = 1;
38                  break;
39              case 1:
40                  mpUkulele.pause();
41                  playing = 0;
42                  break;
43              }
44      }
45  };
46      Button.OnClickListener bDrums = new Button.OnClickListener( ) {
```

Switch decision structure

start( ) method begins the Ukulele song

case 0 starts the music playback

playing assigned to 1 determines that if the user clicks again, case 1 pauses the music

case 1 pauses the music playback

playing assigned to 0 determines that if the user clicks again, case 0 starts the music again

pause( ) method pauses the Ukulele song

Figure 6-27 Switch statements for first onClick method

Using the MediaPlayer Class

The first button changes text while the music is paused or restarted (Figure 6-28).

```
31      Button.OnClickListener bUkulele= new Button.OnClickListener() {
32      @Override
33      public void onClick(View v) {
34          switch(playing) {
35              case 0:
36                  mpUkulele.start();
37                  playing = 1;
38                  button1.setText("Pause Ukulele Song");
39                  break;
40              case 1:
41                  mpUkulele.pause();
42                  playing = 0;
43                  button1.setText("Play Ukulele Song");
44                  break;
45          }
46      }
47  };
```

setText( ) changes the Button text

Figure 6-28 The setText( ) method changes the button control in both case statements

Using the MediaPlayer Class (continued)

Figure 6-29 Coding the button

Changing the Visibility Property Using Code

- The **Visibility property** is the Java property that determines whether a control is displayed on the emulator is
  - Default Visibility property is set to display any control you place on the emulator when the program runs

Code Syntax

To hide the control:	`btnUkulele.setVisibility(View.INVISIBLE);`
To display the control:	`btnUkulele.setVisibility(View.VISIBLE);`

Changing the Visibility Property Using Code

The Drums button is hidden when the music plays and displayed when the music stops (Figure 6-30).

```
31      Button.OnClickListener bUkulele= new Button.OnClickListener() {
32      @Override
33      public void onClick(View v) {
34          switch(playing) {
35              case 0:
36                  mpUkulele.start();
37                  playing = 1;
38                  button1.setText("Pause Ukulele Song");
39                  button2.setVisibility(View.INVISIBLE);
40                  break;
41              case 1:
42                  mpUkulele.pause();
43                  playing = 0;
44                  button1.setText("Play Ukulele Song");
45                  button2.setVisibility(View.VISIBLE);
46                  break;
47          }
48      }
49  };
50      Button.OnClickListener bDrums = new Button.OnClickListener( ) {
51
52          @Override
53          public void onClick(View v) {
54
```

setVisibility( ) method hides Drums button

setVisibility( ) method displays Drums button

Figure 6-30 The setVisibility( ) method changes the visibility of the Button control

Changing the Visibility Property Using Code (continued)

MainActivity.java

```java
package net.androidbootcamp.alohamusic;

import ...

public class MainActivity extends ActionBarActivity {
    Button button1, button2;
    MediaPlayer mpUkulele, mpDrums;
    int playing;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button1 = (Button)findViewById(R.id.btnUkulele);
        button2 = (Button)findViewById(R.id.btnDrums);
        button1.setOnClickListener(bUkulele);
        button2.setOnClickListener(bDrums);
        mpUkulele = new MediaPlayer();
        mpUkulele = MediaPlayer.create(this, R.raw.ukulele);
        mpDrums = new MediaPlayer();
        mpDrums = MediaPlayer.create(this, R.raw.drums);
        playing = 0;
    }
    Button.OnClickListener bUkulele= new Button.OnClickListener() {
    @Override
    public void onClick(View v) {
        switch(playing) {
            case 0:
                mpUkulele.start();
                playing = 1;
                button1.setText("Pause Ukulele Song");
                button2.setVisibility(View.INVISIBLE);
                break;
            case 1:
                mpUkulele.pause();
                playing = 0;
                button1.setText("Play Ukulele Song");
                button2.setVisibility(View.VISIBLE);
                break;
        }
    }
});
```

```java
    Button.OnClickListener bDrums = new Button.OnClickListener( ) {

        @Override
        public void onClick(View v) {
            switch (playing) {
                case 0:
                    mpDrums.start();
                    playing = 1;
                    button2.setText("Pause Drums Song");
                    button1.setVisibility(View.INVISIBLE);
                    break;
                case 1:
                    mpDrums.pause();
                    playing = 0;
                    button2.setText("Play Drums Song");
                    button1.setVisibility(View.VISIBLE);
                    break;

            }
        }

    };
}
```

Figure 6-31 Complete code for MainActivity.java

Summary

- Android apps can show a splash screen that displays program name, brand logo, or author name
- Splash screens open when an app launches
- TextView widgets display a background color or image
- Timers in Java execute a one-time task or perform a continuous process
- Timers must be scheduled to run – timed in milliseconds

Summary

- Each Activity has a life cycle – a series of actions from the beginning of the activity to its end
- Local variables exist within a method and cease to exist when the method is finished
- Variable scope refers to a variable's visibility within a class
- Every Android phone and tablet has a built-in music player
- Music files are typically stored in the res\raw subfolder - In newer versions of Android, you must create the raw subfolder before storing music files

Summary

- The MediaPlayer class provides the methods to control audio playback on an Android device
- The Java property that controls whether a control is displayed on the emulator is the Visible property