Chapter 6: Jam!
Implementing Audio in Android Apps
Objectives

In this chapter, 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 (continued)

- 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
  – Playing Music

Figure 6-1 Eastern Music Android app
Figure 6-2 Music played in the Android app
Implementing Audio (continued)

• 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 approximately 5 seconds when the app opens.
- The next screen opens automatically.
- The Android initializes its resources and loads necessary files while the splash screen is displayed.
Creating a Splash Screen (continued)

- **Adding a Background Image to a TextView Widget**
  - Image is **not** an ImageView Widget
  - Use a TextView widget with a background image

Figure 6-5 splash.xml displays a TextView control
• Creating an Timer
  – A timer in Java can:
    • Execute a one-time task like displaying a 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 List (continued)

• Creating a Timer
  – Code to create a Timer:

        TimerTask task = new TimerTask() {
            @Override
            public void run() {
                // TODO Auto-generated method stub
            }
        }
Creating a Splash Screen (continued)

Figure 6-6 setContentView and Timer task statements

Figure 6-7 run() method
Creating a Splash Screen (continued)

• **Scheduling a Timer**
  – Timers are scheduled in milliseconds
  – 5000 milliseconds = 5 seconds

![Code Snippet]

Figure 6-9 Timer scheduled
Creating a Splash Screen (continued)

• **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
Creating a Splash Screen  

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onCreate()</td>
<td>The onCreate() method begins each Activity. This method also provides a Bundle containing the Activity’s previously frozen state, if it had one.</td>
</tr>
<tr>
<td>onRestart()</td>
<td>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().</td>
</tr>
<tr>
<td>onStart()</td>
<td>If the Activity is hidden, onStart() makes the Activity visible.</td>
</tr>
<tr>
<td>onResume()</td>
<td>The onResume() method is called when the user begins interacting with the Activity. The onResume() method is always followed by onPause().</td>
</tr>
<tr>
<td>onPause()</td>
<td>This method is called when an Activity is about to resume.</td>
</tr>
<tr>
<td>onStop()</td>
<td>This method hides the Activity.</td>
</tr>
<tr>
<td>onDestroy()</td>
<td>This method destroys the Activity. Typically, the finish() method (part of onDestroy()) is used to declare that the Activity is finished; when the next Activity is called, it releases all the resources from the first Activity.</td>
</tr>
</tbody>
</table>

Figure 6-1 methods used in the life cycle of an Activity
Creating a Splash Screen (cont’d)

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

Figure 6-10 Activity life cycle
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)

Figure 6-12 Main class created

Figure 6-13 Adding the Main Activity

Figure 6-14 Intent statement
Designing the main.xml File

Figure 6-15 ImageView and Button controls in main.xml

Figure 6-16 main.xml layout complete
Designing the main.xml File (continued)

• **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*
Designing the main.xml File (continued)

### Figure 6-17 Class variables

```java
package net.androidbootcamp.easternmusic;

import android.app.Activity;
import android.media.MediaPlayer;
import android.widget.Button;

public class Main extends Activity {
    Button btBamboo, btPalace;
    MediaPlayer mpBamboo, mpPalace;
    int playing;
}
```

Button, MediaPlayer, and primitive class variables

### Figure 6-18 onCreate method

```java
package net.androidbootcamp.easternmusic;

import android.app.Activity;   
public class Main extends Activity {   
    Button btBamboo, btPalace;
    MediaPlayer mpBamboo, mpPalace;
    int playing;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

onCreate method opens main.xml
Designing the main.xml File (continued)

Figure 6-19 Button controls referenced
Designing the main.xml File (continued)

Figure 6-20 Inserting the first 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

Figure 6-22 New Folder dialog box

Figure 6-23 Music Files in the raw folder
Using the MediaPlayer Class

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

```java
MediaPlayer mpBamboo = MediaPlayer.create(this, R.raw.bamboo);
```

Figure 6-24 MediaPlayer instance statements
Using the MediaPlayer Class (continued)

- **The MediaPlayer State**

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>start()</td>
<td>Starts media playback</td>
</tr>
<tr>
<td>pause()</td>
<td>Pauses media playback</td>
</tr>
<tr>
<td>stop()</td>
<td>Stops media playback</td>
</tr>
</tbody>
</table>

Table 6-2 Common MediaPlayer states
• Recall the switch statement:

```java
switch(playing) {
    case 0:
        mpBamboo.start();
        playing = 1;
        break;
    case 1:
        mpBamboo.pause();
        playing = 0;
        break;
}
```
Using the MediaPlayer Class (continued)

Figure 6-26 Switch statements for both onClick methods
Changing the Text Property Using Code

`btBamboo.setText("Pause Bamboo Song");`

Figure 6-27 the `setText()` method changes the button control in both case statements
Using the MediaPlayer Class (continued)

- Changing the Visible Property Using Code
  - To hide the control:
    ```java
    btBamboo.setVisibility(View.INVISIBLE);
    ```
  - To display the control:
    ```java
    btBamboo.setVisibility(View.VISIBLE);
    ```
Using the MediaPlayer Class (continued)

- Changing the Visible Property Using Code (cont’)

![Code Snippet]

Figure 6-29 the setVisibility() method changes the visibility of the Button control.
Completed code
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
• 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
• 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