14

In this chapter:
- *What's a Java Applet?*
- *AudioClip Interface*
- *AppletContext Interface*
- *AppletStub Interface*
- *Audio in Applications*

And Then There Were Applets

Although it is not part of the `java.awt` package, the `java.applet` package is closely related. The `java.applet` package provides support for running an applet in the context of a World Wide Web browser. It consists of one class (`Applet`) and three interfaces (`AppletContext`, `AudioClip`, and `AppletStub`). The `Applet` class supports the "applet life cycle" methods (`init()`, `start()`, `stop()`, `destroy()`) that you override to write an applet. `AudioClip` provides support for audio within applets. (Applications use the `sun.audio` package for audio support; `sun.audio` is also covered in this chapter.) The `AppletStub` and `AppletContext` interfaces provide a way for the applet to interact with its run-time environment. Many of the methods of `AppletStub` and `AppletContext` are duplicated in the `Applet` class.

14.1 What's a Java Applet?

Much of the initial excitement about Java centered around applets. Applets are small Java programs that can be embedded within HTML pages and downloaded and executed by a web browser. Because executing code from random Internet sites presents a security risk, Java goes to great lengths to ensure the integrity of the program executing and to prevent it from performing any unauthorized tasks.

An applet is a specific type of Java `Container`. The class hierarchy of an applet is shown in Figure 14-1.

When you are writing an applet, remember that you can use the features of its ancestors. In particular, remember to check the methods of the `Component`, `Container`, and `Panel` classes, which are inherited by the `Applet` class.

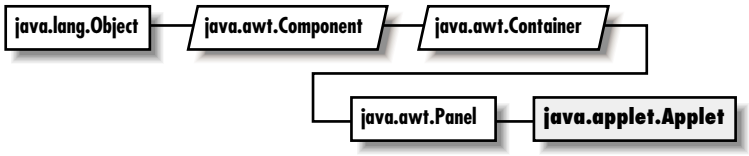

Figure 14–1: Applet class hierarchy

14.1.1 Applet Methods

All the methods of `Applet`, except `setStub()`, either need to be overridden or are methods based on one of the `java.applet` interfaces. The system calls `setStub()` to set up the context of the interfaces. The browser implements the `AppletContext` and `AppletStub` interfaces.

Constructor

public Applet ()

The system calls the `Applet` constructor when the applet is loaded and before it calls `setStub()`, which sets up the applet's stub and context. When you subclass `Applet`, you usually do not provide a constructor. If you do provide a constructor, you do not have access to the `AppletStub` or `AppletContext` and, therefore, may not call any of their methods.

AppletStub setup

public final void setStub (AppletStub stub)

The `setStub()` method of `Applet` is called by the browser when the applet is loaded into the system. It sets the `AppletStub` of the applet to `stub`. In turn, the `AppletStub` contains the applet's `AppletContext`.

Applet information methods

Several methods of `Applet` provide information that can be used while the applet is running.

public AppletContext getAppletContext ()

The `getAppletContext()` method returns the current `AppletContext`. This is part of the applet's stub, which is set by the system when `setStub()` is called.

public URL getCodeBase ()

The `getCodeBase()` method returns the complete URL of the *.class* file that contains the applet. This method can be used with the `getImage()` or the `getAudioClip()` methods, described later in this chapter, to load an image or audio file relative to the *.class* file location.

public URL getDocumentBase ()

The `getDocumentBase()` method returns the complete URL of the *.html* file that loaded the applet. This can be used with the `getImage()` or `getAudioClip()` methods, described later in this chapter, to load an image or audio file relative to the *.html* file.

public String getParameter (String name)

The `getParameter()` method allows you to get run-time parameters from within the `<APPLET>` tag of the *.html* file that loaded the applet. Parameters are defined by HTML `<PARAM>` tags, which have the form:

```
<PARAM name="parameter" value="value>
```

If the `name` parameter of `getParameter()` matches the `name` string of a `<PARAM>` tag, `getParameter()` returns the tag's `value` as a string. If `name` is not found within the `<PARAM>` tags of the `<APPLET>`, `getParameter()` returns `null`. The argument `name` is not case sensitive; that is, it matches parameter names regardless of case. Remember that `getParameter()` always returns a string, even though the parameter values might appear as integers or floating point numbers in the HTML file. In some situations, it makes sense to pass multiple values in a single parameter; if you do this, you have to parse the parameter string manually. Using a `StringTokenizer` will make the job easier.

Enabling your applets to accept parameters allows them to be customized at run-time by the HTML author, without providing the source code. This provides greater flexibility on the Web without requiring any recoding. Example 14-1 shows how an applet reads parameters from an HTML file. It contains three parts: the HTML file that loads the applet, the applet source code, and the output from the applet.

Example 14–1: Getting Parameters from an HTML File

```
<APPLET CODE=ParamApplet WIDTH=100 HEIGHT=100>
<PARAM NAME=one VALUE=1.0>
<PARAM name=TWO value=TOO>
</APPLET>

public class ParamApplet extends java.applet.Applet {
    public void init () {
        String param;
        float one;
        String two;
```

Example 14–1: Getting Parameters from an HTML File (continued)

```
        if ((param = getParameter ("ONE")) == null) {
            one = -1.0f;  // Not present
        } else {
            one = Float.valueOf (param).longValue();
        }
        if ((param = getParameter ("two")) == null) {
            two = "two";
        } else {
            two = param.toUpperCase();
        }
        System.out.println ("One: " + one);
        System.out.println ("Two: " + two);
    }
}

One: 1
Two: TOO
```

public String getAppletInfo ()

The `getAppletInfo()` method lets an applet provide a short descriptive string to the browser. This method is frequently overridden to return a string showing the applet's author and copyright information. How (or whether) to display this information is up to the browser. With *appletviewer*, this information is displayed when the user selects the Info choice under the Applet menu. Neither Netscape Navigator nor Internet Explorer currently display this information.

public String[][] getParameterInfo ()

The `getParameterInfo()` method lets an applet provide a two-dimensional array of strings describing the parameters it reads from `<PARAM>` tags. It returns an array of three strings for each parameter. In each array, the first `String` represents the parameter name, the second describes the data type, and the third is a brief description or range of values. Like `getAppletInfo()`, how (or whether) to display this information is up to the browser. With appletviewer, this information is displayed when the user selects the Info choice under the Applet menu. Neither Netscape Navigator nor Internet Explorer currently display this information. The following code shows how an applet might use `getParameterInfo()` and `getAppletInfo()`:

```
public String getAppletInfo() {
    String whoami = "By John Zukowski (c) 1997";
    return whoami;
}
public String[][] getParameterInfo() {
    String[][] strings = {
        {"parameter1",     "String",      "Background Color name"},
        {"parameter2",     "URL",         "Image File"},
        {"parameter3",     "1-10",        "Number in Series"}
```

```
            };
            return strings;
        }
```

public void showStatus (String message)

The `showStatus()` method displays `message` on the browser's status line, if it has one. Again, how to display this string is up to the browser, and the browser can overwrite it whenever it wants. You should only use `showStatus()` for messages that the user can afford to miss.

public boolean isActive ()

The `isActive()` method returns the current state of the applet. While an applet is initializing, it is not active, and calls to `isActive()` return `false`. The system marks the applet active just prior to calling `start()`; after this point, calls to `isActive()` return `true`.

public Locale getLocale () ★

The `getLocale()` method retrieves the current `Locale` of the applet, if it has one. Using a `Locale` allows you to write programs that can adapt themselves to different languages and different regional variants. If no `Locale` has been set, `getLocale()` returns the default `Locale`. The default `Locale` has a user language of English and no region. To change the default `Locale`, set the system properties `user.language` and `user.region`, or call `Locale.setDefault()` (`setDefault()` verifies access rights with the security manager).*

Applet life cycle

The browser calls four methods of the `Applet` class to execute the applet. These methods constitute the applet's life cycle. The default versions don't do anything; you must override at least one of them to create a useful applet.

public void init ()

The `init()` method is called once when the applet is first loaded. It should be used for tasks that need to be done only once. `init()` is often used to load images or sound files, set up the screen, get parameters out of the HTML file, and create objects the applet will need later. You should not do anything that might "hang" or wait indefinitely. In a sense, `init()` does things that might otherwise be done in an applet's constructor.

public void start ()

The `start()` method is called every time the browser displays the web page containing the applet. `start()` usually does the "work" of the applet. It often starts threads, plays sound files, or does computation. `start()` may also be called when the browser is de-iconified.

---

* For more on the `Locale` class, see *Java Fundamental Classes Reference*, by Mark Grand, from O'Reilly & Associates.

public void stop ()

The `stop()` method is called whenever the browser leaves the web page containing the applet. It should stop or suspend anything that the applet is doing. For example, it should suspend any threads that have been created and stop playing any sound files. `stop()` may also be called when the browser is iconified.

public void destroy ()

The `destroy()` method is called when the browser determines that it no longer needs to keep the applet around—in practice, when the browser decides to remove the applet from its cache or the browser exits. After this point, if the browser needs to display the applet again, it will reload the applet and call the applet's `init()` method. `destroy()` gives the applet a final opportunity to release any resources it is using (for example, close any open sockets). Most applets don't need to implement `destroy()`. It is always a good idea to release resources as soon as they aren't needed, rather than waiting for `destroy()`. There are no guarantees about when `destroy()` will be called; if your browser has a sufficiently large cache, the applet may stay around for a very long time.

Applet-sizing methods

public void resize(int width, int height)

The `resize()` method changes the size of the applet space to `width height`. The browser must support changing the applet space or else the sizing does not change. Netscape Navigator does not allow an applet to change its size; the applet is sized to the region allocated by the `<APPLET>` tag, period.

Because `Applet` is a subclass of `Component`, it inherits the Java 1.1 method `setSize()`, which has the same function.

public void resize (Dimension dim)

This `resize()` method calls the previous version of `resize()` with a width of `dim.width` and a height of `dim.height`.

Images

We have discussed `Image` objects extensively in Chapter 2, *Simple Graphics*, and Chapter 12, *Image Processing*, and used them in many of our examples. When writing an applet, you can use the `getImage()` method directly. In applications, you must go through `Toolkit` (which the following methods call) to get images.

public Image getImage (URL url)

The `getImage()` method loads the image file located at `url`. `url` must be a complete and valid URL. The method returns a system-specific object that sub-

classes `Image` and returns immediately. The `Image` is not loaded until needed, either by `prepareImage()`, `MediaTracker`, or `drawImage()`.

public Image getImage (URL url, String filename)

The `getImage()` method loads the image file located at `url` in `filename`. The applet locates the file relative to the specified URL; that is, if the URL ends with a filename, the applet removes the filename and appends the `filename` argument to produce a new URL. `getImage()` returns a system-specific object that subclasses `Image` and returns immediately. The `Image` is not loaded until needed, either by `prepareImage()`, `MediaTracker`, or `drawImage()`.

In most cases, the `url` argument is a call to `getDocumentBase()` or `getCodeBase()`; most often, image files are located in the same directory as the HTML file, the applet's Java class file, or their own subdirectory.

Audio

Every Java platform is guaranteed to understand Sun's AU file format, which contains a single channel of 8000 Hz μLaw encoded audio data.* Java applets do not require any helper applications to play audio; they use the browser's audio capabilities. You can use an independent application, like Sun's *audiotool*, to control the volume. Of course, the user's workstation or PC needs audio hardware, but these days, it's hard to buy a computer that isn't equipped for audio.

The Java Media Framework API is rumored to provide support for additional audio formats, like Microsoft's *.wav* files or Macintosh/SGI *.aiff* audio files. At present, if you want your Java program to play audio files in other formats, you must first convert the audio file to the *.au* format, using a utility like SOX (Sound Exchange).† Once converted, your Java program can play the resulting *.au* file normally. (If you are interested in more information about audio, look in the *alt.binaries.sounds.d* newsgroup.)

The `Applet` class provides two ways to play audio clips. The first mechanism provides a method to load and play an audio file once:

public void play (URL url)

The `play()` method downloads and plays the audio file located at `url`. `url` must be a complete and valid URL. If `url` is invalid, no sound is played. Some environments throw an exception if the URL is invalid, but not all. Calling `play()` within an applet's `destroy()` method usually has no effect; the applet

* The AU format is explained in the Audio File Format FAQ (version 3.10) located at *ftp://ftp.cwi.nl/pub/audio/index.html* in files *AudioFormats.part1* and *AudioFormats.part2*.

† SOX is available at http://www.spies.com/Sox. The current version of SOX is 10; version 11 is in gamma release. The UNIX source is located in *sox10.tar.gz*, while the DOS executable is *sox10dos.zip*.

and its resources will probably be deallocated before `play()` has time to download the audio file.

public void play (URL url, String filename)

This version of `play()` downloads and plays the audio file located at `url` in the file `filename`. The applet locates the file relative to the specified URL; that is, if the URL ends with a filename, the applet removes the filename and appends the `filename` argument to produce a new URL. If the resulting URL is invalid, no sound is played. Some environments throw an exception if the URL is invalid, but not all.

In most cases, the `url` argument is a call to `getDocumentBase()` or `getCodeBase()`; most often, sound files are located in the same directory as the HTML file or the applet's Java class file. For some reason, you cannot have a double dot (`..`) in the URL of an audio file; you can in the URL of an image file. Putting a double dot in the URL of an audio file raises a security exception in an applet causing `play()` to fail.

The following applet plays an audio file located relative to the HTML file from which the applet was loaded:

```
import java.net.*;
import java.applet.*;
public class audioTest extends Applet {
    public void init () {
        System.out.println ("Before");
        play (getDocumentBase(), "audio/flintstones.au");
        System.out.println ("After");
    }
}
```

The second way to play audio files splits the process into two steps: you get an `AudioClip` object and then play it as necessary. This procedure eliminates a significant drawback to `play()`: if you call `play()` repeatedly, it reloads the audio file each time, making the applet much slower.

public AudioClip getAudioClip (URL url)

The `getAudioClip()` method loads the audio file located at `url`. `url` must be a complete and valid URL. Upon success, `getAudioClip()` returns an instance of a class that implements the `AudioClip` interface. You can then call methods in the `AudioClip` interface (see Section 14.2) to play the clip. If an error occurs during loading (e.g., because the file was not found or the URL was invalid), `getAudioClip()` returns `null`.

`getAudioClip()` sounds similar to `getImage()`, and it is. However, Java currently loads audio clips synchronously; it does not start a separate thread as it does for images. You may want to create a helper class that loads audio clips in a separate thread.

The actual class of the `AudioClip` object depends on the platform you are using; you shouldn't need to know it. If you are curious, the *appletviewer* uses the class `sun.applet.AppletAudioClip`; Netscape Navigator uses the class `netscape.applet.AppletAudioClip`.

public AudioClip getAudioClip (URL url , String filename)

This version of the `getAudioClip()` method loads the audio file located at `url` in the file `filename`. The applet locates the file relative to the specified URL; that is, if the URL ends with a filename, the applet removes the filename and appends the `filename` argument to produce a new URL. If the resulting URL is invalid, the file is not loaded. Upon success, `getAudioClip()` returns an instance of a class that implements the `AudioClip` interface. You can then call methods in the `AudioClip` interface (see Section 14.2) to play the clip. If an error occurs during loading (e.g., because the file was not found or the URL was invalid), `getAudioClip()` returns `null`.

In most cases, the `url` argument is a call to `getDocumentBase()` or `getCodeBase()`; most often, sound files are located in the same directory as the HTML file or the applet's Java class file.

14.2 AudioClip Interface

Once an audio file is loaded into memory with `getAudioClip()`, you use the `AudioClip` interface to work with it.

Methods

Three methods define the `AudioClip` interface. The class that implements these methods depends on the run-time environment; the class is probably `sun.applet.AppletAudioClip` or `netscape.applet.AppletAudioClip`.

If you play an audio clip anywhere within your `Applet`, you should call the `AudioClip` `stop()` method within the `stop()` method of the applet. This ensures that the audio file will stop playing when the user leaves your web page. Stopping audio clips is a must if you call `loop()` to play the sound continuously; if you don't stop an audio clip, the user will have to exit the browser to get the sound to stop playing.

Applets can play audio clips simultaneously. Based upon the user's actions, you may want to play a sound file in the background continuously, while playing other files.

void play ()

The `play()` method plays the audio clip once from the beginning.

void loop ()

The `loop()` method plays the audio clip continuously. When it gets to the end-of-file marker, it resets itself to the beginning.

void stop ()

The `stop()` method stops the applet from playing the audio clip.

14.2.1 Using an AudioClip

The applet in Example 14-2 loads three audio files in the `init()` method. The `start()` method plays Dino barking in the background as a continuous loop. Whenever the browser calls `paint()`, Fred yells "Wilma," and when you click the mouse anywhere, the call to `mouseDown()` plays Fred yelling, "Yabba-Dabba-Doo." If you try real hard, all three can play at once. Before playing any audio clip, the applet makes sure that the clip is not null—that is, that the clip loaded correctly. `stop()` stops all clips from playing; you should make sure that applets stop all audio clips before the viewer leaves the web page.

Example 14–2: AudioClip Usage

```
import java.net.*;
import java.awt.*;
import java.applet.*;
public class AudioTestExample extends Applet{
    AudioClip audio1, audio2, audio3;
    public void init () {
        audio1 = getAudioClip (getCodeBase(), "audio/flintstones.au");
        audio2 = getAudioClip (getCodeBase(), "audio/dino.au");
        audio3 = getAudioClip (getCodeBase(), "audio/wilma.au");
    }
    public boolean mouseDown (Event e, int x, int y) {
        if (audio1 != null)
            audio1.play();
        return true;
    }
    public void start () {
        if (audio2 != null)
            audio2.loop();
    }
    public void paint (Graphics g) {
        if (audio3 != null)
            audio3.play();
    }
    public void stop () {
        if (audio1 != null)
            audio1.stop();
        if (audio2 != null)
            audio2.stop();
```

Example 14–2: AudioClip Usage (continued)

```
            if (audio3 != null)
                audio3.stop();
        }
    }
```

14.3 AppletContext Interface

The `AppletContext` interface provides the means to control the browser environment where the applet is running.

Methods

Some of these methods are so frequently used that they are also provided within the `Applet` class.

public abstract AudioClip getAudioClip (URL url)

The `getAudioClip()` method loads the audio file located at `url`. `url` must be a complete and valid URL. Upon success, `getAudioClip()` returns an instance of a class that implements the `AudioClip` interface. You can then call methods in the `AudioClip` interface (see Section 14.2) to play the clip. If an error occurs during loading (e.g., because the file was not found or the URL was invalid), `getAudioClip()` returns `null`.

public abstract Image getImage (URL url)

The `getImage()` method loads the image file located at `url`. `url` must be a complete and valid URL. The method returns a system-specific object that subclasses `Image` and returns immediately. The `Image` is not loaded until needed. A call to `prepareImage()`, `MediaTracker`, or `drawImage()` forces loading to start.

public abstract Applet getApplet (String name)

The `getApplet()` method fetches the `Applet` from the current HTML page named `name`, which can be the applet's class name or the name provided in the `NAME` parameter of the `<APPLET>` tag. `getApplet()` returns `null` if the applet does not exist in the current context. This method allows you to call methods of other applets within the same context, loaded by the same `ClassLoader`. For example:

```
MyApplet who = (MyApplet)getAppletContext().getApplet("hey");
who.method();
```

TIP

Netscape Navigator 3.0 restricts which applets can communicate with each other. Internet Explorer seems to have a similar restriction. For applets to communicate, they must:

- Have the same `CODEBASE`.
- Have the same or no `ARCHIVES` tag.
- Have `MAYSCRIPT` tags and appear in the same frame; alternatively, neither applet may have a `MAYSCRIPT` tag.

If these conditions are not met and you try to cast the return value of `getApplet()` or `getApplets()` to the appropriate class, either the cast will throw a `ClassCastException`; or nothing will happen, and the method will not continue beyond the point of the failure.

public abstract Enumeration getApplets ()

The `getApplets()` method gathers all the `Applets` in the current context, loaded by the same `ClassLoader`, into a collection and returns the `Enumeration`. You can then cycle through them to perform some operation collectively. For example:

```
Enumeration e = getAppletContext().getApplets();
while (e.hasMoreElements()) {
    Object o = e.nextElement();
    if (o instance of MyApplet) {
        MyApplet a = (Object)o;
        a.MyAppletMethod();
    }
}
```

TIP

If you want communication between applets on one page, be aware that there is no guarantee which applet will start first. Communications must be synchronized by using a controlling class or continual polling.

public abstract void showDocument (URL url)

The `showDocument()` method shows `url` in the current browser window. The browser may ignore the request if it so desires.

public abstract void showDocument (URL url, String frame)

The `showDocument()` method shows `url` in a browser window specified by `frame`. Different `frame` values and the results are shown in Table 14-1. The browser may ignore the request, as *appletviewer* does.

```
try {
    URL u = new URL (getDocumentBase(), (String) file);
    getAppletContext().showDocument (u, "_blank");
} catch (Exception e) {
}
```

Table 14–1: Target Values

Target String	Results
`_blank`	Show `url` new browser window with no name.
`_parent`	Show `url` in the parent frame of the current window.
`_self`	Replace current `url` with `url` (i.e., display in the current window).
`_top`	Show `url` in top-most frame.
`name`	Show `url` in new browser window named `name`.

public abstract void showStatus (String message)

The `showStatus()` method displays `message` on the browser's status line, if it has one. How to display this string is up to the browser, and the browser can overwrite it whenever it wants. You should use `showStatus()` only for messages that the user can afford to miss.

14.4 AppletStub Interface

The `AppletStub` interface provides a way to get information from the run-time browser environment. The `Applet` class provides methods with similar names that call these methods.

Methods

public abstract boolean isActive ()

The `isActive()` method returns the current state of the applet. While an applet is initializing, it is not active, and calls to `isActive()` return `false`. The system marks the applet active just prior to calling `start()`; after this point, calls to `isActive()` return `true`.

public abstract URL getDocumentBase ()

The `getDocumentBase()` method returns the complete URL of the HTML file that loaded the applet. This method can be used with the `getImage()` or `getAudioClip()` methods to load an image or audio file relative to the HTML file.

public abstract URL getCodeBase ()

The `getCodeBase()` method returns the complete URL of the *.class* file that contains the applet. This method can be used with the `getImage()` method or the `getAudioClip()` method to load an image or audio file relative to the *.class* file.

public abstract String getParameter (String name)

The `getParameter()` method allows you to get parameters from `<PARAM>` tags within the `<APPLET>` tag of the HTML file that loaded the applet. The `name` parameter of `getParameter()` must match the name string of the `<PARAM>` tag; `name` is case insensitive. The return value of `getParameter()` is the value associated with `name`; it is always a `String` regardless of the type of data in the tag. If `name` is not found within the `<PARAM>` tags of the `<APPLET>`, `getParameter()` returns `null`.

public abstract AppletContext getAppletContext ()

The `getAppletContext()` method returns the current `AppletContext` of the applet. This is part of the stub that is set by the system when `setStub()` is called.

public abstract void appletResize (int width, int height)

The `appletResize()` method is called by the resize method of the `Applet` class. The method changes the size of the applet space to `width height`. The browser must support changing the applet space; if it doesn't, the size remains unchanged.

14.5 Audio in Applications

The rest of this chapter describes how to use audio in your applications. Because the audio support discussed so far has been provided by the browser, applications that don't run in the context of a browser must use a different set of classes to work with audio. These classes are within the `sun.audio` package. Although the `sun.*` package hierarchy is not necessarily included by other vendors, the `sun.audio` classes discussed here are provided with Netscape Navigator 2.0/3.0 and Internet Explorer 3.0. Therefore, you can use these classes within applets, too. This section ends by developing a `SunAudioClip` class that has an interface similar to the applet's audio interface; you can use it to minimize coding differences between applets and applications.

14.5.1 AudioData

The `AudioData` class holds a clip of 8000 Hz μLaw audio data. This data can be used to construct an `AudioDataStream` or `ContinuousAudioDataStream`, which can then be played with the `AudioPlayer`.

Constructor

public AudioData (byte buffer[])

The `AudioData` constructor accepts a byte array `buffer` and creates an instance of `AudioData`. The buffer should contain 8000 Hz μLaw audio data.

Methods

There are no methods for `AudioData`.

14.5.2 AudioStream

`AudioStream` subclasses `FilterInputStream`, which extends `InputStream`. Using an `InputStream` lets you move back and forth (rewind and fast forward) within an audio file, in addition to playing the audio data from start to finish.

Constructors

public AudioStream (InputStream in) throws IOException

The `AudioStream` constructor has `InputStream in` as its parameter and can throw `IOException` on error. In the following code, we get an input stream by opening a *.au* file. Another common way to construct an `AudioStream` is to use the stream associated with a URL through the URL's `openStream()` method.

```
FileInputStream fis = new FileInputStream ("/usr/openwin/demo/sounds/1.au");
AudioStream audiostream = new AudioStream (fis);
```

or:

```
AudioStream audiostream = new AudioStream (savedUrl.openStream());
```

If you are constructing the audio data yourself, you would use a `ByteArrayInputStream`. Whatever the source of the data, the input stream should provide data in Sun's *.au* format.

Methods

public int read (byte buffer[], int offset, int length) throws IOException

The `read()` method for `AudioStream` reads an array of bytes into `buffer`. `offset` is the first element of `buffer` that is used. `length` is the maximum number of bytes to read. This method blocks until some input is available. `read()` returns the actual number of bytes read. If the end of stream is encountered and no bytes were read, `read()` returns -1. Ordinarily, you `read()` an `AudioStream` only if you want to modify the audio data in some way.

public int getLength()

The `getLength()` method returns the length of the audio data contained within the `AudioStream`, excluding any header information in the file.

public AudioData getData () throws IOException

The `getData()` method of `AudioStream` is the most important and most frequently used. It reads the data from the input stream and creates an `AudioData` instance. As the following code shows, you can create an `AudioStream` and get the `AudioData` with one statement.

```
AudioData audiodata = new AudioStream (aUrl.openStream()).getData();
```

14.5.3 AudioDataStream

Constructors

public AudioDataStream (AudioData data)

This constructor creates an `AudioDataStream` from an `AudioData` object `data`. The resulting `AudioDataStream` is a subclass of `ByteArrayInputStream` and can be played by the `AudioPlayer.start()` method.

Methods

There are no methods for `AudioDataStream`.

14.5.4 ContinuousAudioDataStream

Constructors

public ContinuousAudioDataStream (AudioData data)

This constructor creates a continuous stream of audio from `data`. The resulting `ContinuousAudioDataStream` is a subclass of `AudioDataStream` and, therefore, of `ByteArrayInputStream`. It can be played by `AudioPlayer.start()`; whenever the player reaches the end of the continuous audio data stream, it restarts from the beginning.

Methods

public int read ()

This `read()` method of `ContinuousAudioDataStream` overrides the `read()` method in `ByteArrayInputStream` to rewind back to the beginning of the stream when end-of-file is reached. This method is used by the system when it reads the `InputStream`; it is rarely called directly. `read()` never returns -1 since it loops back to the beginning on end-of-file.

public int read (byte buffer[], int offset, int length)

This `read()` method of `ContinuousAudioDataStream` overrides the `read()` method in `ByteArrayInputStream` to rewind back to the beginning of the stream when end-of-file is reached. This method is used by the system when it reads the `InputStream`; it is rarely called directly. `read()` returns the actual number of bytes read. `read()` never returns -1 since it loops back to the beginning on end-of-file.

14.5.5 AudioStreamSequence

Constructors

public AudioStreamSequence (Enumeration e)

The constructor for `AudioStreamSequence` accepts an `Enumeration e`(normally the elements of a `Vector` of `AudioStreams`) as its sole parameter. The constructor converts the sequence of audio streams into a single stream to be played in order. An example follows:

```
Vector v = new Vector ();
v.addElement (new AudioStream (url1.openStream ());
v.addElement (new AudioStream (url2.openStream ());
AudioStreamSequence audiostream = new AudioStreamSequence (v.elements ());
```

Methods

public int read ()

This `read()` method of `AudioStreamSequence` overrides the `read()` method in `InputStream` to start the next stream when end-of-file is reached. This method is used by the system when it reads the `InputStream` and is rarely called directly. If the end of all streams is encountered and no bytes were read, `read()` returns -1. Otherwise, `read()` returns the character read.

public int read (byte buffer[], int offset, int length)

This `read()` method of `AudioStreamSequence` overrides the `read()` method in `InputStream` to start the next stream when end-of-file is reached. This method is used by the system when it reads the `InputStream` and is rarely called directly. `read()` returns the actual number of bytes read. If the end of all streams is encountered and no bytes were read, `read()` returns -1.

14.5.6 AudioPlayer

The `AudioPlayer` class is the workhorse of the `sun.audio` package. It is used to play all the streams that were created with the other classes. There is no constructor for `AudioPlayer`; it just extends `Thread` and provides `start()` and `stop()` methods.

Variable

public final static AudioPlayer player

`player` is the default audio player. This audio player is initialized automatically when the class is loaded; you do not have to initialize it (in fact, you can't because it is final) or call the constructor yourself.

Methods

public synchronized void start (InputStream in)

The `start()` method starts a thread that plays the `InputStream` `in`. Stream `in` continues to play until there is no more data or it is stopped. If `in` is a `ContinuousAudioDataStream`, the playing continues until `stop()` (described next) is called.

public synchronized void stop (InputStream in)

The `stop()` method stops the player from playing `InputStream` `in`. Nothing happens if the stream `in` is no longer playing or was never started.

14.5.7 SunAudioClip Class Definition

The class in Example 14-3 is all you need to play audio files in applications. It implements the `java.applet.AudioClip` interface, so the methods and functionality will be familiar. The test program in `main()` demonstrates how to use the class. Although the class itself can be used in applets, provided your users have the `sun.audio` package available, it is geared towards application users.

Example 14–3: The SunAudioClip Class

```
import java.net.URL;
import java.io.FileInputStream;
import sun.audio.*;
public class SunAudioClip implements java.applet.AudioClip {
    private AudioData audiodata;
    private AudioDataStream audiostream;
    private ContinuousAudioDataStream continuousaudiostream;
    static int length;
    public SunAudioClip (URL url) throws java.io.IOException {
        audiodata = new AudioStream (url.openStream()).getData();
        audiostream = null;
        continuousaudiostream = null;
    }
    public SunAudioClip (String filename) throws java.io.IOException {
        FileInputStream fis = new FileInputStream (filename);
        AudioStream audioStream = new AudioStream (fis);
        audiodata = audioStream.getData();
        audiostream = null;
        continuousaudiostream = null;
    }
    public void play () {
```

Example 14–3: The SunAudioClip Class (continued)

```
            audiostream = new AudioDataStream (audiodata);
            AudioPlayer.player.start (audiostream);
        }
        public void loop () {
            continuousaudiostream = new ContinuousAudioDataStream (audiodata);
            AudioPlayer.player.start (continuousaudiostream);
        }
        public void stop () {
            if (audiostream != null)
                AudioPlayer.player.stop (audiostream);
            if (continuousaudiostream != null)
                AudioPlayer.player.stop (continuousaudiostream);
        }
        public static void main (String args[]) throws Exception {
            URL url1 = new URL ("http://localhost:8080/audio/1.au");
            URL url2 = new URL ("http://localhost:8080/audio/2.au");
            SunAudioClip sac1 = new SunAudioClip (url1);
            SunAudioClip sac2 = new SunAudioClip (url2);
            SunAudioClip sac3 = new SunAudioClip ("1.au");
            sac1.play ();
            sac2.loop ();
            sac3.play ();
            try {// Delay for loop
                Thread.sleep (2000);
            } catch (InterruptedException ie) {}
            sac2.stop();
        }
    }
```