

Short Guide to Java Applet Programming

Revision 1.1
August 01, 2010

by
Aloysius Indrayanto



(C) 2010 AnemoneSoft.com

This document is multi-licensed under the Creative Commons Attribution Share-Alike (CC-BY-SA) license version 3.0 and the GNU Free Documentation License (GNU FDL) version 1.3 or later.

1. Introduction

Java is a programming language that has a similar syntax with C/C++. However, unlike C/C++, it has a simpler object model and fewer low-level functionality. Java was originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation). The first version of Java was released in 1995. Java applications are compiled into *bytecode* and meant to be run on a Java Virtual Machine (JVM). The same *bytecode* can be run on any JVM that complies with the Java implementation standard. Hence, Java applications can be run in any platform and operating system that are supported by the JVM. Java installation package basically comes in two big flavors:

- The Java Runtime Environment (JRE) that includes the JVM, some tools, and runtime libraries needed to run Java applications. This product is aimed at end users.
- The Java Development Kit (JDK) that includes both the JRE and development tools (compiler, archiver, console, etc.). This product is aimed at Java developers.

In May 2007, Sun released most of its Java technologies under the GNU General Public License. Java standard is now controlled through the Java Community Process.

The word “applet” usually refers to a small application that performs a specific task. However, it can also refer to an application which is embedded in another application to enhance the main application. An applet can run as an independent application as well as within the context of a larger application. The word applet was most likely first used by the scripting language AppleScript in year 1993.

A Java applet is a Java *bytecode* which is embedded in an HTML page. A Java applet can run in a web browser using a JVM or in an AppletViewer (a stand-alone tool for testing applets). Java applets were introduced from the very first version of Java in 1995. Java applets are usually developed by using the Java programming language. However, it is also possible to use other languages (such as Python, Ruby, and Eiffel) that compile into standard-compliant Java *bytecode*. Some advantages of a Java applet are listed below:

- It is cross platform. An applet can run on any browser and operating system that are supported by the JVM.
- JRE is mostly backward-compatible. Hence, an applet developed using an older version of Java usually will still work correctly with newer JRE.

- An applet can distribute works between servers and clients. Hence, reducing the load on the server and make the application more scalable with the number of clients.
- A browser normally load and caches the latest version of an applet. Hence, the server does not need to support all previous versions of the applet because the application will be automatically updated.
- An unsigned applet has no access to the local machine. It can only access the originating server. This would make an applet safer to run than a standalone application in local machine. However, in case access to the local machine is mandatory, a signed applet can be used.
- A Java applet is quite fast because it is compiled into *bytecode* and run by a JVM (not run as an interpreted language).

Some disadvantages of a Java applet are listed below:

- It requires the Java plugin (and hence the installation of the JRE). If an applet requires a newer JRE than one installed in the system (or even a specific version of JRE), the user will need to first download and install the needed JRE before running the applet.
- Installing the Java plugin in some platforms and browsers may be more difficult than it looks.
- Each browser would have its own applet-related bugs.
- The restrictions on unsigned applets may cause difficulties to achieve some tasks.

Please refer to the appendix for the methods to install Java plugin for FireFox and SeaMoneky in Linux. Other browsers (at least Google Chrome and Opera) seem to be able to find the plugin automatically, copy the settings from FireFox, or even provide a GUI interface to select/find the appropriate plugin. In Windows, FireFox should be able to automatically find the Java plugin from the installed JRE.

2. Requirements

A basic knowledge in programming using Java will be needed to understand the topic discussed in this tutorial. A system with an installation of Apache Web Server (HTTPD) version 2.2.x, Java Development Kit (JDK) version 1.6.x (Java Standard Edition 6), and a recent enough browser that can execute Java applet will be needed to run the code snippets.

3. "Hello World" Applet

The code snippet below shows a simple example of how to embed a Java applet in an HTML page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' lang='en' xml:lang='en'>
  <head></head>
  <body>
    <applet width='320' height='240' codebase='HelloWorld/' code='HelloWorld.class'>
    </applet>
  </body>
</html>
```

tut01.html

The codebase attribute basically specify the CLASSPATH of the Java applet. In the example above, all the Java classes will be inside the 'HelloWorld' directory which is under the directory that contains the HTML file. The code attribute specify the main class of the applet (the class to be executed). The code snippet below shows the source code of the applet. Remember that a Java source file must has the same name with the public class defined in it.

```
import java.applet.*;
import java.awt.*;

public class HelloWorld extends Applet {
    // To suppress warning message
    private static final long serialVersionUID = 1L;

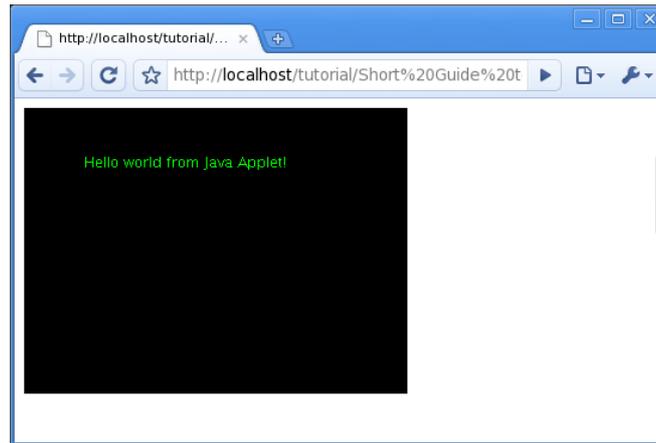
    // Size of the applet
    private int _width, _height;

    // Initialize the applet
    public void init()
    {
        _width = getSize().width;
        _height = getSize().height;
        setBackground(Color.black);
    }

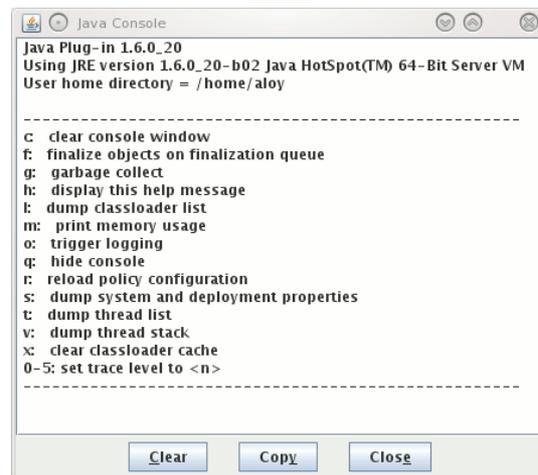
    // Draw the content of the applet
    public void paint(Graphics g)
    {
        g.setColor(Color.green);
        g.drawString("Hello world from Java Applet!", 50, 50);
    }
}
```

HelloWorld.java

Compile the code using the command 'javac -Xlint HelloWorld.java'. The resulting class file 'HelloWorld.class' will be stored in the same directory. When executed in a browser, the output would be similar to:



It is recommended to configure the JDK/JRE to always show the Java console when running an applet (please refer to the appendix). The console can be used to display exception messages, reset the JRE cache, etc. The picture below shows a typical Java console.



Explanation:

- An applet's main class must always extend the Applet class.
- The variable `serialVersionUID` is defined so that the `-Xlint` compiler option will not produce the warning message:

```
warning: [serial] serializable class HelloWorld has no definition  
of serialVersionUID
```

Basically, if there is no real need to serialize the applet object, one can set the variable with any numerical value. It is recommended to always use the `-Xlint` compiler option to enable all Java's recommended warnings so that one can write a cleaner code.

- Our applet class overrides the method `init()` from the Applet class. This method will be called when the browser initializes an applet. In this example, it simply acquire the

size of the applet by using the `getSize()` method and set the background color to black by using the `setBackground()` method.

- Our applet class also overrides the method `paint()` from the `Applet` class. This method will be called whenever the browser wants to redraw the applet. In this example, it only draw a simple text using green color at coordinate (50, 50).

It is possible to develop an applet's main class that can also be run as a stand-alone Java application. This approach can be useful if a developer wants to develop a small utility that needs to be run both from the web and as a local application. However, for online game development, this approach may be not a good idea because it may add unnecessary complication as well as reducing the maintainability and security of the application. The code snippet below shows the needed modifications to the previous code snippet to make it capable of being run as a stand-alone Java application. Note that the applet's main class has been renamed to `HelloWorldST`.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

import java.net.URL;
import java.net.MalformedURLException;

class HelloWorldST_AppletStub implements AppletStub {
    private Applet _applet;

    public HelloWorldST_AppletStub(String argv[], Applet a)
    { _applet = a; }

    public void appletResize(int width, int height)
    { _applet.resize(width, height); }

    public AppletContext getAppletContext()
    { return null; }

    public java.net.URL getCodeBase()
    {
        try { return new URL("file:///"); }
        catch(MalformedURLException e) { return null; }
    }

    public java.net.URL getDocumentBase()
    { return getCodeBase(); }

    public String getParameter(String p)
    { return null; }

    public boolean isActive()
    { return true; }
}

public class HelloWorldST extends Applet {
    ...
    ...
    ...

    // To indicate if the applet has ben run as a stand-alone Java application
    private static boolean _standAlone = false;
}
```

```
...
...
...

// Draw the content of the applet
public void paint(Graphics g)
{
    g.setColor(Color.green);
    if(_standAlone)
        g.drawString("Hello world from Java!", 50, 50);
    else
        g.drawString("Hello world from Java Applet!", 50, 50);
}

// This main() method allows the applet to be run as a stand-alone Java application
static public void main (String argv[])
{
    // Size of the applet
    final int WIDTH = 320;
    final int HEIGHT = 240;

    // Instantiate the applet and applet-stub class
    final Applet applet = new HelloWorldST();
    final AppletStub appletStub = new HelloWorldST_AppletStub(argv, applet);
    applet.setStub(appletStub);
    appletStub.appletResize(WIDTH, HEIGHT);

    // Generate a top-level frame to contain the applet
    Frame frame = new Frame("Hello World");
    frame.addWindowListener(
        new WindowAdapter()
        {
            // Ensure the applet is uninitialized if the frame is closed
            public void windowClosing(WindowEvent event)
            {
                applet.stop();
                applet.destroy();
                System.exit(0);
            }
        }
    );

    // Show the frame
    frame.setResizable(false);
    frame.setVisible(true);

    // Get the frame's insets
    final Insets fi = frame.getInsets();
    final int wo = fi.left + fi.right;
    final int ho = fi.top + fi.bottom;

    // Set the frame's initial position and the frame's size
    frame.setBounds(50, 50, WIDTH + wo, HEIGHT + ho);

    // Add the applet to the frame and start the applet
    frame.add("Center", applet);
    _standAlone = true;
    applet.init();
    applet.start();
}
}
```

HelloWorldST.java

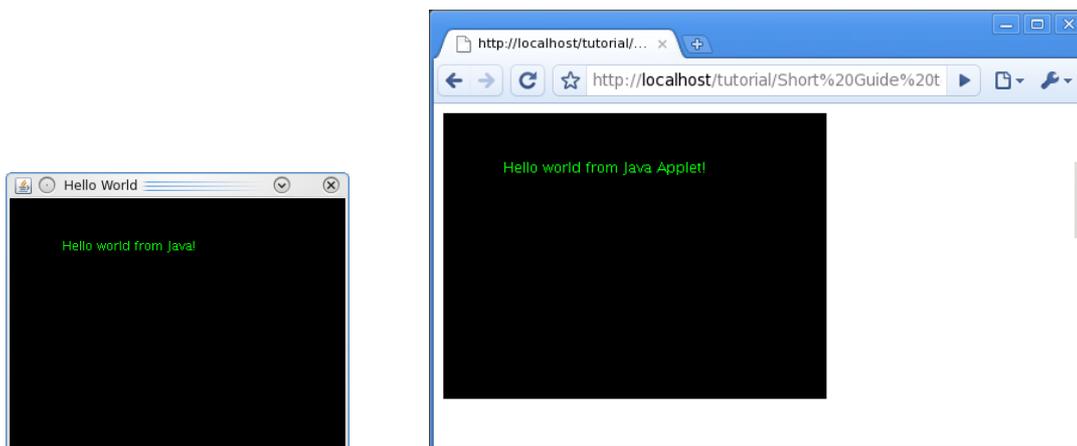
Explanation:

- The first modification is the addition of the HelloWorldST_AppletStub class that implements the AppletStub class. When an applet is run by a browser, it is the browser's

responsibility to supply the implementation of the AppletStub class. In this example, a very basic implementation of the class is used. There cannot be an applet context if an applet is run as a stand-alone application, hence, the `getAppletContext()` method will always return `null`. Due to the applet is run as a local application, the `getCodeBase()` method will always return `'file://'`.

- The second modification is the addition of the `_standAlone` variable to indicate if the application is run as an applet or as a stand-alone application. This variable is used by the modified `paint()` method to draw a different text if the application is run as a stand-alone application.
- The third modification is the addition of the `main()` method in the applet class that allows the applet to be run as a stand-alone application. The jobs of this method are to instantiate the applet and applet stub objects, preparing the containing window, and starting the applet. The implementation of this method would be quite similar for all hybrid application. One would only need to change the required size of the applet (via the `WIDTH` and `HEIGHT` variables) and the title of the window (the `'Hello world'` string in this example).

The applet above can be run as a stand-alone application by using the command `'java HelloWorldST'`. The result would be similar to the pictures below.



An applet can accept parameters from the HTML page by using the `<param/>` tag. It is possible to write a better applet stub class so that one can pass parameters from the command line in case the applet is executed as a stand-alone application. The modified applet stub class is shown below.

```
class HelloWorldST_Ex_AppletStub implements AppletStub {
    private Applet _applet;
    private Hashtable<String, String> _parameters;

    public HelloWorldST_Ex_AppletStub(String argv[], Applet a)
    {
        _applet = a;

        _parameters = new Hashtable<String, String>();
        for(int i = 0; i < argv.length; ++i) {
            try {
                StringTokenizer parser = new StringTokenizer(argv[i], "=");
                String name = parser.nextToken().toString();
                String value = parser.nextToken("\\").toString().substring(1);
                _parameters.put(name, value);
            }
            catch(NoSuchElementException e) {}
        }
    }

    public void appletResize(int width, int height)
    { _applet.resize(width, height); }

    public AppletContext getAppletContext()
    { return null; }

    public java.net.URL getCodeBase()
    {
        try { return new URL("file://"); }
        catch(MalformedURLException e) { return null; }
    }

    public java.net.URL getDocumentBase()
    { return getCodeBase(); }

    public String getParameter(String p)
    { return _parameters.get(p); }

    public boolean isActive()
    { return true; }
}
```

4. Animation in Java Applet – “Bouncing Ball” Applet

The code snippets below show a simple example of how to implement animation with double buffering in Java applet.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>

<html xmlns='http://www.w3.org/1999/xhtml' lang='en' xml:lang='en'>
    <head></head>
    <body>
        <applet width='320' height='240' codebase='BouncingBall/' code='BouncingBall.class'>
        </applet>
    </body>
</html>
```

tut02.html

```
import java.applet.*;
import java.awt.*;

public class BouncingBall extends Applet implements Runnable {
    // To suppress warning message
    private static final long serialVersionUID = 1L;

    // Size of the applet
    private int _width, _height;

    // Backbuffer
    private Image _bbImage;
    private Graphics _bbGraph;

    // Animation thread
    private Thread _threadHandle = null;
    private boolean _threadSuspended = false;

    // Animation data
    private int _xPos = 10, _xSpd = 5;
    private int _yPos = 10, _ySpd = 5;

    // Initialize the applet
    public void init()
    {
        _width = getSize().width;
        _height = getSize().height;
        setBackground(Color.black);

        _bbImage = createImage(_width, _height);
        _bbGraph = _bbImage.getGraphics();
    }

    // Start the applet
    public void start()
    {
        // Create the thread if it is not yet exist
        if(_threadHandle == null) {
            _threadHandle = new Thread(this);
            _threadSuspended = false;
            _threadHandle.start();
        }

        // Resume the thread if it is currently suspended
        else if(_threadSuspended) {
            _threadSuspended = false;
            synchronized(this) {
                notify();
            }
        }
    }

    // Stop the applet
    public void stop()
    { _threadSuspended = true; }

    // Draw the content of the applet
    public void paint(Graphics g)
    { update(g); }

    // Update the content of the applet
    public void update(Graphics g)
    { g.drawImage(_bbImage, 0, 0, this); }

    // Thread procedure
    public void run()
    {
        try {
            while (true) {
                // Check if the thread should suspend itself
                if(_threadSuspended) {
                    synchronized(this) {
                        while(_threadSuspended) wait();
                    }
                }
            }
        }
    }
}
```

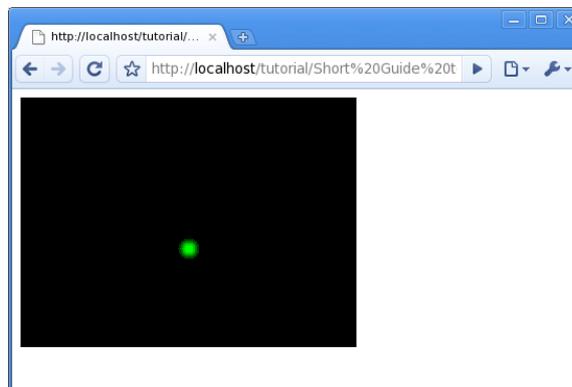
```

    }
}
// Draw the ball
_bbGraph.setColor(Color.black);
_bbGraph.fillRect(0, 0, _width, _height);
_bbGraph.setColor(new Color(0, 64, 0));
_bbGraph.fillOval(_xPos, _yPos, 20, 20);
_bbGraph.setColor(new Color(0, 128, 0));
_bbGraph.fillOval(_xPos + 2, _yPos + 2, 16, 16);
_bbGraph.setColor(new Color(0, 207, 0));
_bbGraph.fillOval(_xPos + 4, _yPos + 4, 12, 12);
_bbGraph.setColor(new Color(0, 255, 0));
_bbGraph.fillOval(_xPos + 6, _yPos + 6, 8, 8);
// Update the ball's position
_xPos += _xSpd;
_yPos += _ySpd;
if(_xPos < 0) { _xPos = 0; _xSpd = -_xSpd; }
if(_xPos > _width - 20) { _xPos = _width - 20; _xSpd = -_xSpd; }
if(_yPos < 0) { _yPos = 0; _ySpd = -_ySpd; }
if(_yPos > _height - 20) { _yPos = _height - 20; _ySpd = -_ySpd; }
// Repaint and sleep
repaint();
_threadHandle.sleep(50);
}
}
catch(InterruptedException e) {}
}
}

```

BouncingBall.java

Compile the code using the command 'javac -Xlint BouncingBall.java'. The result would be similar to the picture below.



Explanation:

- The animation is implemented using thread. Hence, our class will need to both extend the Applet class and implement the Runnable class.
- The init() method creates a new back buffer image and then acquiring its graphics context. Animation with double buffering can be done by drawing only to a back buffer image/graphics context, and after all the drawing processes finishes, blit the back buffer to the front buffer (the applet).

- The `start()` method initializes a new animation thread in case the thread is not exist yet. If the thread already exists, it simply resume the thread. This approach is necessary because browsers may suspend an applet's thread if the user navigate away from the page.
- The `stop()` method just set the `_threadSuspended` flag to `true`.
- The `paint()` method simply forwards the call to the `update()` method.
- The `update()` method will simply blit the back buffer to the front buffer. It is necessary to blit from within the `update()` method and *not* from the `paint()` method to prevent flicker.
- The `run()` method is the heart of the animation. It basically contains an infinite loop with an exception handler.
 - First, it checks if it will need to suspend the animation thread by checking if the `_threadSuspended` flag has been set to `true`.
 - If the thread does not need to be suspended, it will clear the back buffer and draw several circles to form a ball.
 - Next, it will update the ball's position.
 - It will then order the applet to repaint itself by calling the `repaint()` method.
 - Finally, it will sleeps for about 50 milliseconds to delay the animation. Note that in a real application, one would want to use a variable sleeping time.

4.1. Adding Sound Effect

All unsigned applets cannot access local files. Therefore from this section onwards, you will need to host the HTML and applet classes in a web server. It is recommended to always use a local (personal) web server when developing an applet-based application so that one can be sure that the applet will be executed as if it is in a real deployment environment (the internet).

The first step to add sound effects is to create an audio player class that utilizes features from `javax.sound.sampled`. However, it is important to understand that the current Java sound library (Java Standard Edition 6) has some limitations:

- In some platforms, it opens the system audio device exclusively. No other application can play audio until the first Java application closes the system audio device.
- A developer would need to write a custom mixer to play multiple audio files at the same time (for mixing the sound effects).

The code snippet below shows a simple audio player class that can only play audio files which are encoded in plain PCM (Pulse-Code Modulation). The method to write a custom mixer will not be discussed in this tutorial.

```
class AudioPlayer implements Runnable {
    private static final int INIT_BUFFER_SIZE = 1024 * 16;
    private static final int READ_BUFFER_SIZE = 1024 * 128;

    private byte[] _buff = null;
    private Mixer _mixer = null;
    private AudioFormat _af = null;
    private DataLine.Info _dli = null;
    private Thread _th = null;

    public AudioPlayer(String urlAudio)
    {
        try {
            // Open the audio
            URL url = new URL(urlAudio);
            AudioInputStream ais = AudioSystem.getAudioInputStream(url.openStream());

            // Get mixer, format, and data line
            _mixer = AudioSystem.getMixer(null);
            _af = ais.getFormat();
            _dli = new DataLine.Info(SourceDataLine.class, _af);

            // Allocate the initial audio buffer
            _buff = new byte[INIT_BUFFER_SIZE];

            // Read the audio data
            byte[] readBuff = new byte[READ_BUFFER_SIZE];
            int totCount = 0;
            while(true) {
                // Read and check for end of stream
                int readCount = ais.read(readBuff, 0, readBuff.length);
                if(readCount <= 0) break;
                // Resize the audio buffer (if needed)
                if(totCount + readCount > _buff.length) {
                    // Allocate a new buffer
                    byte[] newBuff = new byte[_buff.length +
                        Math.max(INIT_BUFFER_SIZE, readCount)];

                    // Copy data
                    System.arraycopy(_buff, 0, newBuff, 0, _buff.length);
                    // Swap buffer
                    _buff = newBuff;
                }
                // Copy data to the audio buffer
                System.arraycopy(readBuff, 0, _buff, totCount, readCount);
                totCount += readCount;
            }

            // Truncate the audio buffer to fit the full size of the audio stream
            if(totCount < _buff.length) {
                // Allocate a new buffer
                byte[] newBuff = new byte[totCount];
                // Copy data
                System.arraycopy(_buff, 0, newBuff, 0, totCount);
                // Swap buffer
                _buff = newBuff;
            }
        }
        catch(MalformedURLException e) {
            System.out.printf("MalformedURLException\n");
            System.out.printf("%s\n", e.getMessage());
        }
        catch(IOException e) {
            System.out.printf("IOException\n");
            System.out.printf("%s\n", e.getMessage());
        }
    }
}
```

```
        catch(UnsupportedAudioFileException e) {
            System.out.printf("UnsupportedAudioFileException\n");
            System.out.printf("%s\n", e.getMessage());
        }
    }

    public void run()
    {
        SourceDataLine line = null;

        try {
            line = (SourceDataLine) _mixer.getLine(_dli);
            line.open(_af);
            line.start();
            line.write(_buff, 0, _buff.length);
            line.drain();
        }
        catch(LineUnavailableException e) {
            System.out.printf("LineUnavailableException\n");
            System.out.printf("%s\n", e.getMessage());
        }
        catch(Exception e) {
            System.out.printf("Exception\n");
            System.out.printf("%s\n", e.getMessage());
        }

        if(line != null) line.close();
        line = null;
        _th = null;
    }

    public void play()
    {
        if(_buff.length <= 0 || _th != null) return;

        _th = new Thread(this);
        _th.start();
    }
}
```

Explanation:

- On initialization the class' constructor opens the given URL as an audio stream.
- It then will acquire the system's default mixer, the format of the audio stream, and construct a data-line information object related to the format of the audio stream.
- Next, it will prepare a buffer and copy the entire audio stream to the buffer. If the final buffer size is larger than the whole length of the audio data, the buffer will be truncated. It is important to copy the audio data to local buffer so that the application will not need to re-request the audio data from the server.
- The run() method is meant to be run in a separated thread to feed the audio device with data. It basically will open a line, start the line, write data into the line, wait until the playback completed, and finally close the line.
- The play() method simply creates a new thread and start it (only if the audio is properly loaded and the thread is not yet running).

The code snippet below shows the needed modifications to the original bouncing ball applet to make it capable of playing sound effect.

```
import java.applet.*;
import java.awt.*;

import java.lang.Math;

import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;

import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.Mixer;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.UnsupportedAudioFormatException;
import javax.sound.sampled.LineUnavailableException;

class AudioPlayer implements Runnable {
    ...
    ...
    ...
}

public class BouncingBallWS extends Applet implements Runnable {
    ...
    ...
    ...

    // Animation data
    private int _xPos = 10;
    private int _yPos = 10;
    private int _xSpd = 5;
    private int _ySpd = 5;
    private AudioPlayer _ap = null;

    // Initialize the applet
    public void init()
    {
        _width = getSize().width;
        _height = getSize().height;
        _codeBase = getCodeBase().toString();

        setBackground(Color.black);

        _bbImage = createImage(_width, _height);
        _bbGraph = _bbImage.getGraphics();

        _ap = new AudioPlayer(_codeBase + "../cling.wav");
    }

    ...
    ...
    ...

    // Thread procedure
    public void run()
    {
        try {
            while (true) {
                ...
                ...
                ...

                // Update the ball's position and play audio (if needed)
                boolean pAud = false;
                _xPos += _xSpd;
                _yPos += _ySpd;
            }
        }
    }
}
```

```
        if(_xPos < 0) { _xPos = 0; _xSpd = -_xSpd; pAud = true; }
        if(_xPos > _width - 20) { _xPos = _width - 20; _xSpd = -_xSpd; pAud = true; }
        if(_yPos < 0) { _yPos = 0; _ySpd = -_ySpd; pAud = true; }
        if(_yPos > _height - 20) { _yPos = _height - 20; _ySpd = -_ySpd; pAud = true; }
        if(pAud) _ap.play();
        // Repaint and sleep
        repaint();
        _threadHandle.sleep(50);
    }
}
catch(InterruptedException e) {}
}
```

BouncingBallWS.java

Explanation:

- The first modification is adding more import statements to load support for network, I/O, and sound.
- The second modification is a new private member variable `_ap` that will be initialized in the constructor of the applet's main class. It is important to generate the audio stream URL using the string returned by the `getCodeBase()` method rather than hard-coding the URL. This would allow specifying path relative to the CLASSPATH of the applet (the codebase attribute of the `<applet></applet>` tag).
- The final modification is in the animation thread procedure. Anytime the ball hits the edge of the screen (applet's window), a flag is set so that an audio effect will be played.

4.2. Making It a Stand-Alone Application

It is quite simple to convert the previous code snippet so that it can be run both as an applet and as a stand-alone application. There are only three modifications that need to be done:

- Adding an applet stub class.
- Modifying the `AudioPlayer` class constructor to open the audio URL as a local file if the URL protocol is `'file://'`.
- Adding a `main()` method to the applet's main class.

Please refer to `'HelloWorldST.java'` for the details of the applet stub class and the `main()` method. The code snippet below shows the necessary modifications to the previous code snippet to make it capable of being run as a stand-alone Java application. Note that the applet's main class has been renamed to `BouncingBallWSST`.

```
class BouncingBallWSST_AppletStub implements AppletStub {
    ...
    ...
    ...
}

class AudioPlayer implements Runnable {
    ...
    ...
    ...

    public AudioPlayer(String locAudio)
    {
        try {
            // Open the audio
            URL url = new URL(locAudio);
            AudioInputStream ais = null;
            if(url.getProtocol() == "file")
                ais = AudioSystem.getAudioInputStream(new File(url.getPath()));
            else
                ais = AudioSystem.getAudioInputStream(url.openStream());

            ...
            ...
            ...
        }
    }

    public class BouncingBallWSST extends Applet implements Runnable {
        ...
        ...
        ...

        // To indicate if the applet has ben run as a stand-alone Java application
        private static boolean _standAlone = false;

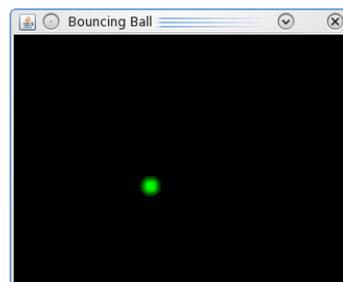
        ...
        ...
        ...

        // This main() method allows the applet to be run as a stand-alone Java application
        static public void main (String argv[])
        {
            ...
            ...
            ...

            // Generate a top-level frame to contain the applet
            Frame frame = new Frame("Bouncing Ball");
            ...
            ...
            ...
        }
    }
}
```

BouncingBallWSST.java

The result would be similar to the picture below.



4.3. Playing Ogg Vorbis and MP3 Audio

Playing Ogg Vorbis and MP3 audio streams have been made easy by using plugins/libraries from <http://www.tritonius.org/plugins.html> and <http://www.jcraft.com/jorbis>. The code snippet below shows the necessary modification to the `AudioPlayer` class so that it can load compressed audio stream. The name of the audio file is now also obtained from the `clingAudioFile` applet parameter.

```
...
...
...

public AudioPlayer(String urlAudio)
{
    try {
        // Open the audio
        URL url = new URL(urlAudio);
        AudioInputStream sis = AudioSystem.getAudioInputStream(url.openStream());
        AudioFormat sfm = sis.getFormat();

        // Get mixer, format, and data line
        _mixer = AudioSystem.getMixer(null);
        _af = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
            sfm.getSampleRate(),
            16, // 16 bits per samples
            sfm.getChannels(),
            sfm.getChannels() * 2, // 16 bits are 2 bytes
            sfm.getSampleRate(),
            false // Little endian
        );
        _dli = new DataLine.Info(SourceDataLine.class, _af);

        // Convert the audio
        AudioInputStream ais = AudioSystem.getAudioInputStream(_af, sis);

        ...
        ...
        ...
    }

    ...
    ...
    ...

    public void init()
    {
        ...
        ...
        ...

        _ap = new AudioPlayer(_codeBase + getParameter("clingAudioFile"));
    }

    ...
    ...
    ...
}
```

BouncingBallWS_Decode.java

Decompressing/decoding a compressed audio format to plain PCM can simply be done by chaining two audio streams. The first one is used to read the audio data from the URL, the second one is used to convert (decompress/decode) the audio stream to the desired format.

The code snippets below demonstrate the usage of the archive attribute to load the needed libraries and the <param/> tag to define the name of the audio file. The libraries needed to play Ogg Vorbis audio files are a bit different than the libraries needed to play MP3 audio files.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>

<html xmlns='http://www.w3.org/1999/xhtml' lang='en' xml:lang='en'>

  <head></head>

  <body>
    <applet width    = '320'
            height   = '240'
            codebase = 'BouncingBallWS_Decode/'
            code     = 'BouncingBallWS_Decode.class'
            archive  = 'jorbis-0.0.17.jar,
                      tritonus_share-0.3.6.jar, tritonus_jorbis-0.3.6.jar'>
      <param name='clingAudioFile' value='../cling.ogg' />
    </applet>
    <div>
      Using Ogg Vorbis plugin/library from<a href='http://www.tritonus.org/plugins.html'>
      http://www.tritonus.org/plugins.html</a>
      and <a href='http://www.jcraft.com/jorbis/'>http://www.jcraft.com/jorbis</a>.
    </div>
  </body>

</html>
```

tut02ws_ovb.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>

<html xmlns='http://www.w3.org/1999/xhtml' lang='en' xml:lang='en'>

  <head></head>

  <body>
    <applet width    = '320'
            height   = '240'
            codebase = 'BouncingBallWS_Decode/'
            code     = 'BouncingBallWS_Decode.class'
            archive  = 'javalayer.jar,
                      tritonus_share-0.3.6.jar, tritonus_mp3-0.3.6.jar'>
      <param name='clingAudioFile' value='../cling.mp3' />
    </applet>
    <div>
      Using MP3 plugin/library from<a href='http://www.tritonus.org/plugins.html'>
      http://www.tritonus.org/plugins.html</a>.
    </div>
  </body>

</html>
```

tut02ws_mp3.html

There seems to be bugs either in the libraries or in the JRE (or both). Only one audio format can be played by the same instance of JRE. If you run the Ogg Vorbis code snippet first and then run the MP3 code snippet, the second one will not run (it will produce error), and *vice versa*. One would need to restart the browser each time one wants to play different audio formats.

5. Using the <embed></embed> Tag

It is easy (and recommended) to replace the <applet></applet> tag with the newer <embed></embed> tag as shown in the code snippet below.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
<html xmlns='http://www.w3.org/1999/xhtml' lang='en' xml:lang='en'>
  <head></head>
  <body>
    <embed type          = 'application/x-java-applet'
          width         = '320'
          height        = '240'
          codebase      = 'BouncingBallWS_Decode/'
          code          = 'BouncingBallWS_Decode.class'
          archive       = 'jorbis-0.0.17.jar,
                        tritonus_share-0.3.6.jar, tritonus_jorbis-0.3.6.jar'
          pluginspage   = 'http://java.com/en/download/index.jsp'
          clingAudioFile = '../cling.ogg'>
    </embed>
    <div>
      Using Ogg Vorbis plugin/library from <a href='http://www.tritonus.org/plugins.html'>
      http://www.tritonus.org/plugins.html</a>
      and <a href='http://www.jcraft.com/jorbis/'>http://www.jcraft.com/jorbis</a>.
    </div>
  </body>
</html>
```

tut02ws_ovb_embed.html

If an applet needs a specific version of JRE (for example Java Standard Edition 5), the type attribute can be extended to 'application/x-java-applet;version=1.5'. The pluginspage attribute should point to the URL from where users can download the needed plugin.

It is clear that using the <embed></embed> tag is a bit more complicated. However, it provides more features (the type and pluginspace attributes). The <embed></embed> tag is part of the new HTML 5 standard. Unless, there is a need to support very old browsers, using the <embed></embed> tag is strongly recommended. Try it for yourself to convert all the code snippets in this tutorial to use the <embed></embed> tag.

Appendix

```
=====
Installing Java Plugin for x86 Linux
=====
```

- * For Firefox 2.0 or later and SeaMonkey 1.0 or later.
- * Firefox 3.1 and later need JRE 1.6.0_10 or later.

```
-----
For Older Firefox
-----
```

Create a symbolic link to libjavaplugin_oji.so (located in the 'plugin/i386/ns7' directory of your JRE installation) in your Mozilla plugins directory. Examples:

```
cd /usr/lib/firefox-3.6/plugins
ln -s /usr/java/jre1.6.0_20/plugin/i386/ns7/libjavaplugin_oji.so .

cd /usr/lib/firefox-3.6/plugins
ln -s /usr/java/jdk1.6.0_20/jre/plugin/i386/ns7/libjavaplugin_oji.so .

cd /usr/lib/firefox-3.6/plugins
ln -s /usr/java/latest/jre/plugin/i386/ns7/libjavaplugin_oji.so .
```

Please adjust the Firefox and Java plugins directories as needed.

```
-----
For Firefox 3.1 and later
-----
```

Create a symbolic link to libnpp2.so (located in the 'lib/i386' directory of your JRE installation) in your Mozilla plugins directory. Examples:

```
cd /usr/lib/firefox-3.6/plugins
ln -s /usr/java/jre1.6.0_20/lib/i386/libnpp2.so .

cd /usr/lib/firefox-3.6/plugins
ln -s /usr/java/jdk1.6.0_20/jre/lib/i386/libnpp2.so .

cd /usr/lib/firefox-3.6/plugins
ln -s /usr/java/latest/jre/lib/i386/libnpp2.so .
```

Please adjust the Firefox and Java plugins directories as needed.

```
=====
Installing Java Plugin for x86_64 Linux
=====
```

- * Available from JRE 1.6.0_12 or later.
- * For Firefox 3.0 or later and pre-release versions of SeaMonkey 2.0.

Create a symbolic link to libnpp2.so (located in the 'lib/amd64' directory of your JRE installation) in your Mozilla plugins directory. Examples:

```
cd /usr/lib64/firefox-3.6/plugins
ln -s /usr/java/jre1.6.0_20/lib/amd64/libnpp2.so .

cd /usr/lib64/firefox-3.6/plugins
ln -s /usr/java/jdk1.6.0_20/jre/lib/amd64/libnpp2.so .

cd /usr/lib64/firefox-3.6/plugins
ln -s /usr/java/latest/jre/lib/amd64/libnpp2.so .
```

Please adjust the Firefox and Java plugins directories as needed.

=====
Configuring Java to Always Show the Console
=====

- * Run any of the commands:
 /usr/java/jre1.6.0_20/bin/jcontrol
 /usr/java/latest/bin/jcontrol
 /usr/java/jdk1.6.0_20/bin/jcontrol
- * Go to the 'Advanced' tab.
- * Expand the 'Java Console' section.
- * Select 'Show Console'.
- * Click 'Apply' and then 'OK'.

References

[http://en.wikipedia.org/wiki/Java_\(programming_language\)](http://en.wikipedia.org/wiki/Java_(programming_language)), July 30, 2010

http://en.wikipedia.org/wiki/Java_Development_Kit, July 30, 2010

http://en.wikipedia.org/wiki/Java_applet, July 30, 2010

<http://en.wikipedia.org/wiki/Applet>, July 30, 2010

<http://en.wikipedia.org/wiki/AppleScript>, July 30, 2010

<http://www.oracle.com/technetwork/java/index.html>, July 30, 2010

<http://download.oracle.com/javase/tutorial/java/index.html>, July 30, 2010

<http://profs.logti.etsmtl.ca/mmcguffin/learn/java>, July 30, 2010