# 

Animation in Java Applets

**Summary**

Many Java applets perform animation, whether it's the classic, cartoon-style animation of Duke waving, program-generated lines such as flowing sine waves, or simply moving static images across the screen. No comprehensive, direct-manipulation tools exist (yet) for creating Java animations. So it's up to the applet programmer to do all the hard work.



**By Arthur van Hoff with Kathy Walrath**

This article describes how to implement animation using the Java applet API. It describes commonly used techniques and gives a simple example to illustrate each technique. Here's what you'll read about:

* [Basic Animation Techniques](http://www.javaworld.com/jw-03-1996/animation/index.html#basicTechniques)
  + [Using a Thread](http://www.javaworld.com/jw-03-1996/animation/index.html#thread)
  + [Keeping a Constant Frame Rate](http://www.javaworld.com/jw-03-1996/animation/index.html#frameRate)
  + [Painting Each Frame](http://www.javaworld.com/jw-03-1996/animation/index.html#painting)
  + [Generating Graphics](http://www.javaworld.com/jw-03-1996/animation/index.html#graphics)
* [Avoiding Excessive Flashing](http://www.javaworld.com/jw-03-1996/animation/index.html#flashing)
  + [Overriding the update() Method](http://www.javaworld.com/jw-03-1996/animation/index.html#update)
  + [Double Buffering](http://www.javaworld.com/jw-03-1996/animation/index.html#doubleBuffer)
* [Using Images](http://www.javaworld.com/jw-03-1996/animation/index.html#images)
  + [Moving an Image Across the Screen](http://www.javaworld.com/jw-03-1996/animation/index.html#movingImage)
  + [Displaying a Sequence of Images](http://www.javaworld.com/jw-03-1996/animation/index.html#imageSequence)
  + [Using MediaTracker to Avoid Incremental Display](http://www.javaworld.com/jw-03-1996/animation/index.html#MediaTracker)
* [Adding Sound](http://www.javaworld.com/jw-03-1996/animation/index.html#sound)
* [Tips for Loading Images Faster](http://www.javaworld.com/jw-03-1996/animation/index.html#imageLoading)
  + [Using an Image Strip](http://www.javaworld.com/jw-03-1996/animation/index.html#imageStrip)
  + [Inter-Frame Compression Using Flic](http://www.javaworld.com/jw-03-1996/animation/index.html#flic)
* [Animation Tools](http://www.javaworld.com/jw-03-1996/animation/index.html#tools)
* [Conclusion](http://www.javaworld.com/jw-03-1996/animation/index.html#thatsallfolks)

## Basic Animation Techniques

Many forms of animation are possible in Java. What all of them have in common is that they create some kind of motion on the screen by drawing successive frames at a relatively high speed (usually about 10-20 times per second).

We will start by creating a simple template applet for doing animations and slowly elaborate it until we arrive at a fairly complete applet.

**Using a Thread**

To update the screen multiple times per second, you need to create a new Java thread that contains an animation loop. The animation loop is responsible for keeping track of the current frame and for requesting periodic screen updates. To implement a thread, you must either create a subclass of [Thread](http://java.sun.com/JDK-1.0/api/java.lang.Thread.html) or adhere to the [Runnable](http://java.sun.com/JDK-1.0/api/java.lang.Runnable.html) interface.

A common mistake is to put the animation loop in the [**paint()**](http://java.sun.com/JDK-1.0/api/java.awt.Component.html#paint) method of an applet. Doing so will have strange side effects because it holds up the main AWT thread, which is in charge of all drawing and event handling.

As an example I have written a small template applet, called Example1Applet, that illustrates the general outline of an animation applet. Example1Applet shows how to create a thread and call the [**repaint()**](http://java.sun.com/JDK-1.0/api/java.awt.Component.html#repaint) method at fixed intervals. The number of frames per second is specified by passing in an applet parameter. Here is an example of what you would put in your HTML document:

<applet code=Example1Applet.class width=200 height=200>  
 <param name=fps value=20>  
 </applet>

[**Here is Example1Applet.**](http://www.javaworld.com/jw-03-1996/animation/Example1Applet.html)

**Note:**

This applet doesn't actually draw anything on the screen yet. Drawing to the screen is explained later. Note also that the applet destroys its animation thread whenever the user leaves the page (which results in the applet's **stop()** method being called). This ensures that the applet won't waste CPU time while its page isn't visible.

**Keeping a Constant Frame Rate**

In the example above, the applet simply sleeps for a fixed amount of time between frames. This has the drawback that you sometimes wait too long. To get 10 frames per second you should not wait 100 milliseconds between frames, because you lose some time just running the thread.

The following applet, Example2Applet, shows how to keep better time. It simply computes the correct delay between frames by keeping track of the starting time. It computes the estimated required delay between frames based on the current time.

[**Here is Example2Applet.**](http://www.javaworld.com/jw-03-1996/animation/Example2Applet.html)

**Painting Each Frame**

What remains is to paint each frame. In the previous examples, we call **repaint()** for each frame, which causes the applet's [**paint()**](http://java.sun.com/JDK-1.0/api/java.awt.Component.html#paint) method to be called. The Example3Applet has a **paint()** method that draws the number of the current frame to the screen.

[**Here is Example3Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example3Applet.html)

**Note:**

If you specify the frame rate to be very high (say 100 frames per second), the **run()** method will call **repaint()** 100 times per second. However, this will not always result in 100 calls to **paint()** per second because when you issue repaint request too quickly they will be collapsed into a single screen update. This is why we keep track of the current frame number in the **run()** method rather then in the **paint()** method.

**Generating Graphics**

Now let's animate something that is a little harder to draw. The Example4Applet draws a combination of sine waves. For each x coordinate, it draws a short vertical line. All these lines together form a simple graph that changes for each frame. Unfortunately, you will find that this approach causes a lot of flashing. We'll explain the cause of the flashing and some remedies in the next section.

[**Here is Example4Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example4Applet.html)

## Avoiding Excessive Flashing

The flashing you see in Example4Applet has two causes: painting each frame takes too long (due to the amount of computation that is required during the repaint) and the entire background is cleared before **paint()** is called. While the computation of the next frame is going on, the user is seeing the background of the animation.

This short time between the clearing of the background and the painting of the sine wave is seen as a flash. On some platforms like the PC, the flashing is more obvious then it is on X Windows. The reason is that the X Windows graphics are buffered, which makes the flash a little shorter.

You can reduce flashing greatly using two simple tricks: implementing the **update()** method and using double buffering (sometimes known as *using a backbuffer*).

**Overriding the update() Method**

When the AWT receives a repaint request for an applet, it calls the applet's **update()** method. By default, the **update()** method clears the applet's background and then calls the **paint()** method. By overriding the **update()** method to include the drawing code that used to be in the **paint()** method, we avoid having the applet's entire area cleared with every repaint.

Now that the background is no longer cleared automatically, we need to do it ourselves in the **update()** method. We can now erase each vertical line of the graph individually before drawing the new line, eliminating the flashing completely. This effect is shown in Example5Applet.

[**Here is Example5Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example5Applet.html)

**Note:**

Whenever you override the **update()** method, you still need to implement **paint()**. This is because the **paint()** method is called directly by the AWT drawing system whenever "damage" occurs to the applet's drawing area -- for example, when a window obscuring part of the applet's drawing area is removed from the screen. Your **paint()** implementation can simply call **update()**.

**Double Buffering**

Another way of reducing the flashing between frames is to use double buffering. This technique is used in many animation applets.

The general principle is that you create an offscreen image, you draw a frame into the image, and then you slap the entire image onto the screen with one call to **drawImage()**. The advantage is that most of the drawing is done offscreen. The final painting of the offscreen image onto the screen is usually much more efficient than painting the frame directly to the screen.

The sine wave applet with double buffering is shown in Example6Applet. You will see that the animation is pretty smooth and you don't need any special tricks when drawing the frame. The only disadvantage is that you have to allocate an offscreen image that is as large as the drawing area. If the drawing area is very large, this may demand quite a lot of memory.

[**Here is Example6Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example6Applet.html)

**Note:**

When you use double buffering, you need to override the **update()** method, since you don't want the applet's background to be cleared before you paint the frame. (You clear the background yourself by drawing to the offscreen image.)

## Using Images

Now we'll rewrite the **paintFrame()** method with a method that animates some images. This adds some minor complications to the problem. Images are rather large and they are loaded incrementally. It can take a long time for images to be fully drawn, especially when you are loading them over a slow connection. This is the reason why the **drawImage()** method takes a fourth argument, an ImageObserver object. The image observer is an object that is notified when more of the image data has arrived. To get the images we use the **getImage()** method.

**Moving an Image Across the Screen**

This first image-animating applet, Example7Applet, uses the following two images:

world.gif:  car.gif: 

The world image is used as the background, and the car image is drawn on top of it twice, creating an animation of two cars racing across the world.

[**Here is Example7Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example7Applet.html)

**Displaying a Sequence of Images**

Example8Applet shows how to create an animation using separate images for each frame. Here are the 10 frames that are being used:

T1.gif:  T2.gif:  T3.gif:  T4.gif:  T5.gif: 

T6.gif:  T7.gif:  T8.gif:  T9.gif:  T10.gif: 

We are still using double buffering to eliminate flashing. The reason is that each image that we are rendering is partially transparent, and we therefore need to erase each frame before drawing the next. This would cause flashing without double buffering.

[**Here is Example8Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example8Applet.html)

**Note:**

When displaying sequences of images, you have to be careful to align the images correctly. The easiest way is to make sure that the images are all the same size and can be drawn at the same position. If that isn't the case, your applet will have to draw each frame at a different offset.

**Using MediaTracker to Avoid Incremental Display**

When a Java program loads an image, it can display the image before the image is completely loaded. The user sees the image being rendered first incompletely, and then incrementally more and more completely as the image is loaded. This incremental display gives the user feedback (improving perceived performance) and lets the program easily perform other tasks while the image is loading.

Where animation is concerned, incremental image display can be useful for background images, but it can be very distracting when used for the animated images. It is therefore sometimes desirable to wait until the entire animation is loaded before displaying it.

You can use [Jim Graham's](http://java.sun.com/people/flar/) [MediaTracker](http://java.sun.com/JDK-1.0/api/java.awt.MediaTracker.html) class to track the downloading of images, delaying the animation display until the entire set of images is fully downloaded. [Example9Applet](http://www.javaworld.com/jw-03-1996/animation/Example9Applet.java) shows how to use the MediaTracker class to download images for the waving Duke animation.

[**Here is Example9Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example9Applet.html)

## Adding Sound

It's easy to add sound to an animation. You can use the **getAudioClip()** method to get an AudioClip object. Later, you can play the clip either as a continuous loop or as a single sound. Example10Applet shows how to play a continuous background sound as well as a repetitive sound during the animation.

[**Here is Example10Applet in action, followed by a code listing.**](http://www.javaworld.com/jw-03-1996/animation/Example10Applet.html)

**Note:**

When playing a continuous sound you must remember to stop it when the user leaves the page (i.e., do it in your applet's **stop()** method).

**Another Note:**

Continuous audio can be very annoying. It is a good idea to provide the user with a way to turn off the audio without leaving the page. You can provide a button, or simply turn off the audio when the user clicks in the applet.

## Tips for Loading Images Faster

An animation that uses many images will take a long time to download. This is mainly due to the fact that a new HTTP connection is made for every image file, and making a connection can take several seconds even when there is plenty of bandwidth.

In this section, we'll tell you about two image formats your applet can use to make downloading images faster.

**Using an Image Strip**

You can improve downloading performance by using a single image containing several frames of animation. You can render a single frame out of the image by using the **clipRect()** operator. Below is an example of an image strip that is used in the UnderConstruction applet.

jack.gif: 

The applet creates a drilling effect by not erasing the previous frames. The background is cleared only every so often.

[**Here is UnderConstruction in action, with a link to its source code.**](http://www.javaworld.com/jw-03-1996/animation/UnderConstruction.html)

**Inter-Frame Compression Using Flic**

If you really want to improve the downloading performance of an animation consisting of multiple frames, then you have to use some form of inter-frame compression. I have found two sites that support the Flic compression format.

* [Streaming Animation from DimensionX](http://www.dnx.com/dnx/StreamingAnimation/)
* [Ray Waldin's Flic Player](http://www.primenet.com/~rwaldin/FlicIP.html)

## Animation Tools

At this moment (January 1996), few tools are available to help you create Java-powered animations. The best tool that I could find is [DimensionX](http://www.dnx.com/)'s [The Easy Animator (TEA)](http://www.dnx.com/scott/jam/) (previously known as JAM). It lets you create animations interactively. We'd like to encourage developers to write more tools for the creating of animations in Java.

If you have a few ready-made images to display, you can use the [Animator applet.](http://java.sun.com/applets/applets/Animator/) Animator has many parameters that let you specify continuous sounds, frame-specific sounds, individual frame timing and positions, a startup image, frame ordering, and so on.

You should also check out the [Gamelan Animation](http://www.gamelan.com/Gamelan.animation.html) page to find many applets that use animation.

## Conclusion

I hope this article will help applet developers write more and better animation applets. I also hope that better tools will become available soon. 

**About the authors**

**** [Arthur van Hoff](http://java.sun.com/people/avh/) was, until recently, a Senior Staff Engineer at [Sun Microsystems](http://www.sun.com/) and has been involved in the development of the Java language since 1993. He is the author of the first Java compiler written entirely in Java. He recently left Sun to form a new company together with Sami Shaio, Kim Polese, and Jonathan Payne. The new company will focus on building Java applications. Reach him at [arthur.vanhoff@javaworld.com](http://www.javaworld.com/javaworld/cgi-bin/jw-mailto.cgi?avh@netcom.com+/javaworld/jw-03-1996/jw-03-animation.html+avh)

Kathy Walrath is a technical writer at [Sun Microsystems](http://www.sun.com/). She has been part of the Java team since 1993. Currently, she is working with Mary Campione on [*The Java Tutorial: Object-Oriented Programming for the Internet*](http://java.sun.com/tutorial/), an applet-enhanced tutorial for learning the Java language, applet programming, and Java GUI programming. Besides being available online, *The Java Tutorial* will also be published this summer as part of the Addison-Wesley Java Series. Reach her at [kathy.walrath@javaworld.com](http://www.javaworld.com/javaworld/cgi-bin/jw-mailto.cgi?kwalrath@eng.sun.com+/javaworld/jw-03-1996/jw-03-animation.html+walrath)

**What did you think of this article?**

-Excellent! -Okay -Poor  -Too long -Just right -Too short

Comments:

   





If you have problems with this magazine, contact [webmaster@javaworld.com](http://www.javaworld.com/javaworld/cgi-bin/jw-mailto.cgi?webmaster@javaworld.com+/javaworld/jw-03-1996/jw-03-animation.html+webmaster)

URL: http://www.javaworld.com/javaworld/jw-03-1996/jw-03-animation.html

Last updated: 15 February 1996