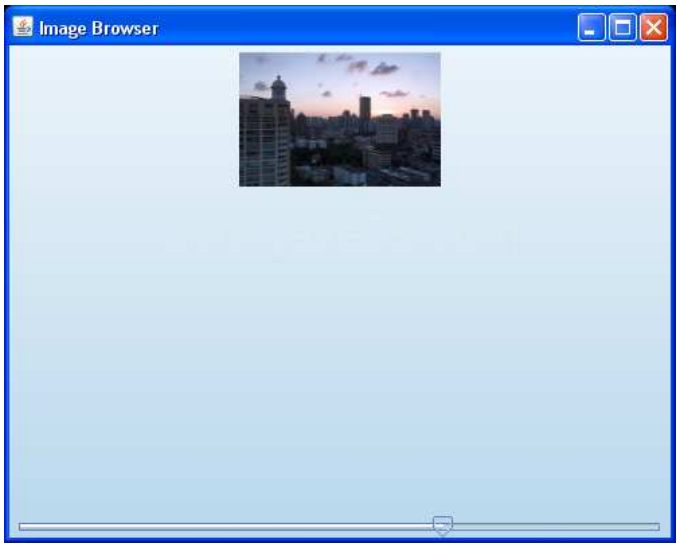
Image Zooming




```
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSlider;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import org.jdesktop.animation.timing.Animator;
import org.jdesktop.animation.transitions.Effect;
import org.jdesktop.animation.transitions.EffectsManager;
import org.jdesktop.animation.transitions.EffectsManager.TransitionType;
import org.jdesktop.animation.transitions.ScreenTransition;
import org.jdesktop.animation.transitions.TransitionTarget;
import org.jdesktop.animation.transitions.effects.CompositeEffect;
import org.jdesktop.animation.transitions.effects.Move;
import org.jdesktop.animation.transitions.effects.Scale;
//import org.jdesktop.tools.io.FileTreeWalk;
//import org.jdesktop.tools.io.FileTreeWalker;
//import org.jdesktop.tools.io.UnixGlobFileFilter;
```

```
import java.io.FileFilter;
import java.io.File;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
/*
 * ImageBrowser.java
 *
 * Created on May 3, 2007, 3:11 PM
 *
 * Copyright (c) 2007, Sun Microsystems, Inc
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials provided
 *     with the distribution.
 *   * Neither the name of the TimingFramework project nor the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * This demo of the AnimatedTransitions library uses a layout manager
 * to assist in setting up the next screen that the application
 * transitions to.
 *
 * The slider in the window controls the picture thumbnail size. The
 * standard FlowLayout manager organizes the pictures according to
 * the thumbnail sizes. The transition animates the change from
 * one thumbnail size to the next.
 *
 * @author Chet
 */
public class ImageBrowser extends JComponent
        implements TransitionTarget, ChangeListener {

    private static final int SLIDER_INCREMENT = 50;
    int numPictures = 40;
    JLabel label[];
    Animator animator = new Animator(500);
    ScreenTransition transition = new ScreenTransition(this, this, animator);
    Dimension newSize = new Dimension();
    List<ImageHolder> images = new ArrayList<ImageHolder>();
    static int currentSize = 50;
    GradientPaint bgGradient = null;
    int prevHeight = 0;
    static JSlider slider = new JSlider(1, 400 / SLIDER_INCREMENT,
            1 + currentSize / SLIDER_INCREMENT);
    static int numImages = 0;

    /** Creates a new instance of ImageBrowser */
    public ImageBrowser() {
        setOpaque(true);
        animator.setAcceleration(.1f);
        animator.setDeceleration(.4f);
        setLayout(new FlowLayout());
        loadImages();
        label = new JLabel[images.size()];
        // For each image:
        // - set the icon at the current thumbnail size
        // - create/set a custom effect that will move/scale the
        // images. Note that the main reason for the custom effect
        // is that scaling effects typically redraw the actual component
        // instead of using image tricks. In this case, image tricks are
        // just fine. So the custom effect is purely an optimization here.
        for (int i = 0; i < images.size(); ++i) {
            label[i] = new JLabel();
            label[i].setIcon(new ImageIcon(images.get(i).getImage(currentSize)));
            add(label[i]);
            Effect move = new Move();
            Effect scale = new Scale();
            CompositeEffect comp = new CompositeEffect(move);
            comp.addEffect(scale);
            comp.setRenderComponent(false);
            EffectsManager.setEffect(label[i], comp, TransitionType.CHANGING);
        }
    }

    /**
     * Paints a gradient in the background of this component
     */
    @Override
    protected void paintComponent(Graphics g) {
```

```
        if (getHeight() != prevHeight) {
            prevHeight = getHeight();
            bgGradient = new GradientPaint(0, 0,
                    new Color(0xEBF4FA), 0, prevHeight, new Color(0xBBD9EE));
        }
        ((Graphics2D)g).setPaint(bgGradient);
        g.fillRect(0, 0, getWidth(), prevHeight);
    }

    /**
     * Loads all images found in the directory "images" (which therefore must
     * be found in the folder in which this app runs).
     */
    private void loadImages() {
        //try {
            //File imagesDir = new File("images");
            //FileTreeWalker walker = new FileTreeWalker(imagesDir,
              //      new UnixGlobFileFilter("*.jpg"));
            //walker.walk(new FileTreeWalk() {
              //  public void walk(File path) {
                    //numImages++;
                    try {
                        BufferedImage image = ImageIO.read(ImageBrowser.class.getResource("shanghai.jpg"));
                        images.add(new ImageHolder(image));
                    } catch (Exception e) {
                        System.out.println("Problem loading images: " + e);
                    }
                //}
            //});
        //} catch (Exception e) {
          //  System.out.println("Problem loading images: " + e);
        //}
    }

    /**
     * TransitionTarget implementation: The setup for the next screen entails
     * merely assigning a new icon to each JLabel with the new thumbnail
     * size
     */
    public void setupNextScreen() {
        for (int i = 0; i < images.size(); ++i) {
            label[i].setIcon(new ImageIcon(images.get(i).getImage(currentSize)));
        }
        // revalidation is necessary for the LayoutManager to do its job
        revalidate();
    }

    /**
     * This method handles changes in slider state, which can come from either
     * mouse manipulation of the slider or right/left keyboard events. This
     * event changes the current thumbnail size and starts the transition.
     * We will then receive a callback to setupNextScreen() where we set up
     * the GUI according to this new thumbnail size.
     */
    public void stateChanged(ChangeEvent ce) {
        currentSize = slider.getValue() * 25;
        if (!transition.getAnimator().isRunning()) {
            transition.start();
        }
    }

    private static void createAndShowGUI() {
  JFrame f = new JFrame("Image Browser");
        f.setLayout(new BorderLayout());
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  f.setSize(500, 400);
  ImageBrowser component = new ImageBrowser();
  f.add(component, BorderLayout.CENTER);
        f.add(slider, BorderLayout.SOUTH);
        slider.setBackground(new Color(0xBBD9EE));
        slider.addChangeListener(component);
  f.setVisible(true);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (InstantiationException ex) {
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            ex.printStackTrace();
        } catch (UnsupportedLookAndFeelException ex) {
            ex.printStackTrace();
        }
  Runnable doCreateAndShowGUI = new Runnable() {
      public void run() {
    createAndShowGUI();
      }
  };
  SwingUtilities.invokeLater(doCreateAndShowGUI);
    }
}

/**
 * This is a utility class that holds our images at various scaled
 * sizes. The images are pre-scaled down by halves, using the progressive
```

```
 * bilinear technique. Thumbnails from these images are requested
 * from this class, which are created by down-scaling from the next-largest
 * pre-scaled size available.
 */
class ImageHolder {
    private List<BufferedImage> scaledImages = new ArrayList<BufferedImage>();
    private static final int MIN_SIZE = 50;

    /**
     * Given any image, this constructor creates and stores down-scaled
     * versions of this image down to some MIN_SIZE
     */
    ImageHolder(BufferedImage originalImage) {
        int imageW = originalImage.getWidth();
        int imageH = originalImage.getHeight();
        scaledImages.add(originalImage);
        BufferedImage prevImage = originalImage;
        while (imageW > MIN_SIZE && imageH > MIN_SIZE) {
            imageW = imageW >> 1;
            imageH = imageH >> 1;
            BufferedImage scaledImage = new BufferedImage(imageW, imageH,
                    prevImage.getType());
            Graphics2D g2d = scaledImage.createGraphics();
            g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
            g2d.drawImage(prevImage, 0, 0, imageW, imageH, null);
            g2d.dispose();
            scaledImages.add(scaledImage);
        }
    }

    /**
     * This method returns an image with the specified width. It finds
     * the pre-scaled size with the closest/larger width and scales
     * down from it, to provide a fast and high-quality scaed version
     * at the requested size.
     */
    BufferedImage getImage(int width) {
        for (BufferedImage scaledImage : scaledImages) {
            int scaledW = scaledImage.getWidth();
            // This is the one to scale from if:
            // - the requested size is larger than this size
            // - the requested size is between this size and
            //   the next size down
            // - this is the smallest (last) size
            if (scaledW < width || ((scaledW >> 1) < width) ||
                    (scaledW >> 1) < MIN_SIZE) {
                if (scaledW != width) {
                    // Create new version scaled to this width
                    // Set the width at this width, scale the
                    // height proportional to the image width
                    float scaleFactor = (float)width / scaledW;
                    int scaledH = (int)(scaledImage.getHeight() *
                            scaleFactor + .5f);
                    BufferedImage image = new BufferedImage(width,
                            scaledH, scaledImage.getType());
                    Graphics2D g2d = image.createGraphics();
                    g2d.setRenderingHint(
                            RenderingHints.KEY_INTERPOLATION,
                            RenderingHints.VALUE_INTERPOLATION_BILINEAR);
                    g2d.drawImage(scaledImage, 0, 0,
                            width, scaledH, null);
                    g2d.dispose();
                    scaledImage = image;
                }
                return scaledImage;
            }
        }
        // shouldn't get here
        return null;
    }
}
/**
 * Copyright (c) 2006, Sun Microsystems, Inc
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials provided
 *     with the distribution.
 *   * Neither the name of the Harvester project nor the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
 */


class UnixGlobFileFilter implements FileFilter {
    private Pattern pattern;

    public UnixGlobFileFilter(String filter) {
        pattern = Pattern.compile(globToRegex(filter));
    }

    public boolean accept(File file) {
        String path = file.getName();
        Matcher matcher = pattern.matcher(path);
        return matcher.matches();
    }

    private String globToRegex(String glob) {
        char c = '\0';
        boolean escape = false;
        boolean enclosed = false;
        StringBuffer buffer = new StringBuffer(glob.length());

        for (int i = 0; i < glob.length(); i++) {
            c = glob.charAt(i);

            if (escape) {
                buffer.append('\\');
                buffer.append(c);
                escape = false;
                continue;
            }

            switch (c) {
                case '*':
                    buffer.append('.').append('*');
                    break;
                case '?':
                    buffer.append('.');
                    break;
                case '\\':
                    escape = true;
                    break;
                case '.':
                    buffer.append('\\').append('.');
                    break;
                case '{':
                    buffer.append('(');
                    enclosed = true;
                    break;
                case '}':
                    buffer.append(')');
                    enclosed = false;
                    break;
                case ',':
                    if (enclosed)
                        buffer.append('|');
                    else
                        buffer.append(',');
                    break;
                default:
                    buffer.append(c);
            }
        }
        return buffer.toString();
    }
}
/**
 * Copyright (c) 2006, Sun Microsystems, Inc
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials provided
 *     with the distribution.
 *   * Neither the name of the Harvester project nor the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


class FileTreeWalker {
    private File path;
```

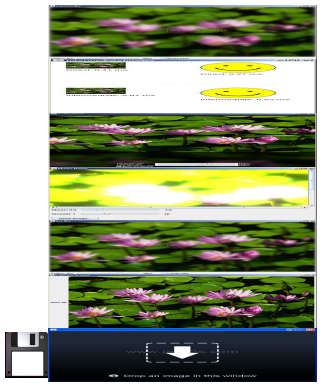
```
    private static final FileFilter directoryFilter = new FileFilter() {
        public boolean accept(File pathname) {
            return pathname.isDirectory();
        }
    };
    private FileFilter filter;

    public FileTreeWalker(File path) throws IOException {
        this(path, new FileFilter() {
            public boolean accept(File pathname) {
                return pathname.isFile();
            }
        });
    }

    public FileTreeWalker(File path, FileFilter filter) throws IOException {
        if (path == null || !path.exists() || path.isFile()) {
            throw new IOException("Path " + path + " is not a valid directory.");
        }
        this.path = path;
        this.filter = filter;
    }

    public void walk(FileTreeWalk walk) {
        walkDirectory(walk, path);
    }

    private void walkDirectory(FileTreeWalk walk, File dir) {
        File[] files = dir.listFiles(filter);
        for (File file : files) {
            walk.walk(file);
        }

        File[] dirs = dir.listFiles(directoryFilter);
        for (File subDir : dirs) {
            walkDirectory(walk, subDir);
        }
    }
}
/**
 * Copyright (c) 2006, Sun Microsystems, Inc
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials provided
 *     with the distribution.
 *   * Neither the name of the Harvester project nor the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

interface FileTreeWalk {
    public void walk(File path);
}
```

 Filthy-Rich-Clients-ImageZooming.zip(228 k)

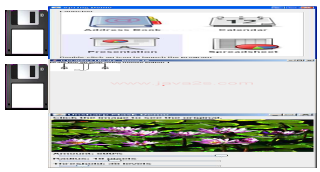
Related examples in the same category

1. Create an image that does not support transparency
2. Create an image that supports transparent pixels
3. Create an image that supports arbitrary levels of transparency
4. Creating a buffered image using Component.createImage().
5. Creating a Buffered Image from an Image

6. Drawing on a Buffered Image
7. If the buffered image supports transparency
8. Converting a Buffered Image (BufferedImage) from an Image
9. Getting and Setting Pixels in a Buffered Image
10. Scaling a Buffered Image
11. Shearing a Buffered Image
12. Translating a Buffered Image
13. Rotating a Buffered Image
14. Flipping a Buffered Image
15. Flip the image horizontally
16. Flip the image vertically and horizontally, equivalent to rotating the image 180 degrees
17. 2D Image Draw
18. Transform image
19. Creating a Image Zoomer using Graphics2D
20. Gaussian Blur Demo



21. Intermediate Images
22. Image reflection
23. Bloom Demo
24. Box Blur Demo
25. Brightness Increase Demo
26. Blur an Image
27. Blur our image: Blur means an unfocused image
28. A reflected image: effect makes an illusion as if the image was reflected in water
29. Enlarge Image With Animation



30. Simple image handling and drawing interaction
31. Unsharp Mask Demo
32. Create a grayscale image with Java 2D tools
33. A 3x3 kernel that embosses an image.
34. A 3x3 kernel that blurs an image.
35. A 3x3 kernel that sharpens an image.
36. Embossing a Buffered Image
37. Brighten the image by 30%
38. Darken the image by 10%
39. Extend RGBImageFilter to create ColorFilter class
40. Extend RGBImageFilter to create AlphaFilter class
41. Enlarging an image by pixel replication
42. Shrinking an image by skipping pixels