```
/**
 * This object is a tool to use when doing some basic graphics in
 * Java. For basic use, initialize the GraphicsWindow and then call
 * the <code>getPen()</code> or <code>getGraphics2D()</code> methods,
 * to obtain the <code>Graphics2D</code> object to use to draw. When
 * you're done, be sure to call <code>finalize()</code> so that
 * everything you drew is visible.
 *
 * Version 2.0 introduced the use of &ldquo;double buffering&rdquo;,
 * for flicker-free animation. To use, define the new object with the
 * new special constructor. Draw to the object in the same way as
 * before (though nothing will appear yet), and then call
 * <code>flip()</code> to flip the invisible back buffer to the
 * front. Repeat the process when the next frame of animation is
 * ready.
 *
 * @author             Adam Smith
 * @version            2.0
 */

import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.*;
import javax.imageio.ImageIO;

public class GraphicsWindow extends Frame {
        private BufferedImage image, backImage;
        private int leftOffset, topOffset;

        /**
         * Create a new <code>GraphicsWindow</code> to hold graphical
         * output, without double buffering. Use this constructor if you
         * aren't animating anything.
         * @param title the title of the window on your desktop
         * @param width the width of the window's content
         * @param height the height of the window's content
         * @param x the initial x-location of the window on the desktop
         * @param y the initial y-location of the window on the desktop
         * @since 1.0
         */

        public GraphicsWindow(String title, int width, int height, int x, int
y) {
                this(title, null, width, height, x, y, false);
        }

        /**
         * Create a new <code>GraphicsWindow</code> to hold graphical output,
with possible double buffering for animation.
         * @param title the title of the window on your desktop
         * @param width the width of the window's content
         * @param height the height of the window's content
         * @param x the initial x-location of the window on the desktop
         * @param y the initial y-location of the window on the desktop
         * @param doDoubleBuffering if true, double buffering is activated
```

```
         * @since 2.0
         */

        public GraphicsWindow(String title, int width, int height, int x, int
y, boolean doDoubleBuffering) {
                this(title, null, width, height, x, y, doDoubleBuffering);
        }

        /**
         * Create a new <code>GraphicsWindow</code> to hold graphical output,
with possible double buffering for animation.
         * @param title the title of the window on your desktop
         * @param icon the icon to use for this window
         * @param width the width of the window's content
         * @param height the height of the window's content
         * @param x the initial x-location of the window on the desktop
         * @param y the initial y-location of the window on the desktop
         * @param doDoubleBuffering if true, double buffering is activated
         * @since 2.0
         */

        public GraphicsWindow(String title, Image icon, int width, int height,
int x, int y, boolean doDoubleBuffering) {
                super(title); // call the Frame constructor, on which this
object is based

                // if an image was passed, use it as the icon
                if (icon != null) setIconImage(icon);

                setLocation(x, y);

                // set the window to close, when the user hits the close
button
                addWindowListener(new WindowAdapter() {public void
windowClosing(WindowEvent e) {destroy();}});

                // calculate offsets for border, title bar, etc.
                setVisible(true);
                setResizable(false);
                Insets insets = getInsets();
                leftOffset = insets.left;
                topOffset = insets.top;

                // set the size to be big enough for all that
                setSize(width+leftOffset+insets.right,
height+topOffset+insets.bottom);

                // define the images inside the window (with no double
buffering, they are aliases)
                image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
                if (doDoubleBuffering) backImage = new BufferedImage(width,
height, BufferedImage.TYPE_INT_RGB);
                else backImage = image;

                // paint a black rectangle, as the first image
                paintBackground(Color.BLACK);
```

```
                // paint the whole thing
                repaint();
        }

        /**
         * Copies the image fron the front buffer to the back buffer.
         * @since 2.0
         */

        public void copyBack() {
                backImage.createGraphics().drawImage(image, null, 0, 0);
        }

        /**
         * Eliminates this <code>GraphicsWindow</code> object, erasing it from
the
         * screen.
         * @since 2.0
         */

        public void destroy() {
                image = backImage = null;
                dispose();
        }

        /**
         * Make sure that everything you drew is properly showing. This is
         * really just an alias for <code>repaint()</code>. If you are
         * using double buffering, use <code>flip()</code> instead.
         * @since 2.0
         */

        public void finalize() {
                repaint();
        }

        /**
         * Flip the image, so the back buffer is showing. If there is no
double buffering, effectively just calls repaint().
         * @since 2.0
         */

        public void flip() {
                BufferedImage swap = backImage;
                backImage = image;
                image = swap;
                repaint();
        }

        /**
         * Acquire a <code>Graphics2D</code> object to use to paint.
         * @return the <code>Graphics2D</code> object
         * @since 1.0
         */

        public Graphics2D getGraphics2D() {
```

```
                return backImage.createGraphics();
        }

        /**
         * Alias for {@link #getGraphics2D()}. Acquire a
<code>Graphics2D</code> object to use to paint.
         * @return the <code>Graphics2D</code> object
         * @since 1.0
         */

        public Graphics2D getPen() {
                return getGraphics2D();
        }

        /**
         * Get the height of the window's active area.
         * @return the height
         * @since 2.0
         */

        public int getImageHeight() {
                return image.getHeight();
        }

        /**
         * Get the width of the window's active area.
         * @return the width
         * @since 2.0
         */

        public int getImageWidth() {
                return image.getWidth();
        }

        /**
         * Static utility to load an image file as a
<code>BufferedImage</code>, to draw later.
         * @param filename the name of the image file (JPEG, PNG, GIF, etc.)
         * @return the loaded image
         * @since 1.0
         */

        public static BufferedImage loadImage(String filename) {
                BufferedImage image;
                try {
                        return ImageIO.read(new File(filename));
                }
                catch (IOException e) {
                        System.err.println("Had a problem loading " +filename+
".");
                        return null;
                }
        }

        /**
         * Static utility to load an image file as a 1D array of
<code>BufferedImage</code>s.
```

```
         * It loads the file, chops it into several images, and returns the
set as an array.
         * @param filename the name of the image file (JPEG, PNG, GIF, etc.)
         * @param tileSize the width of each subimage
         * @return the array of loaded images
         * @since 1.0
         */

        public static BufferedImage[] loadImageAsTiles(String filename, int
tileSize) {
                int i;

                BufferedImage master;
                BufferedImage[] tiles;
                int tileHeight;

                // load up the big image
                master = loadImage(filename);

                // get a bunch of tiles, that are subimages of the big one
                tileHeight = master.getHeight();
                tiles = new BufferedImage[master.getWidth() / tileSize];
                for (i=0; i<tiles.length; i++) {
                        tiles[i] = master.getSubimage(i*tileSize, 0, tileSize,
tileHeight);
                }

                return tiles;
        }

        /**
         * Static utility to load an image file as a 2D array of
<code>BufferedImage</code>s.
         * It loads the file, chops it into several images, and returns the
set as an array.
         * @param filename the name of the image file (JPEG, PNG, GIF, etc.)
         * @param tileWidth the width of each subimage
         * @param tileHeight the height of each subimage
         * @return the 2D array of loaded images (row, then column)
         * @since 1.0
         */

        public static BufferedImage[][] loadImageAsTiles(String filename, int
tileWidth, int tileHeight) {
                int i, j;

                BufferedImage master;
                BufferedImage[][] tiles;

                // load up the big image
                master = loadImage(filename);

                // get a bunch of tiles, that are subimages of the big one
                tiles = new BufferedImage[master.getHeight() /
tileHeight][master.getWidth() / tileWidth];
                for (i=0; i<tiles.length; i++) {
                        for (j=0; j<tiles[i].length; j++) {
```

```
                                tiles[i][j] = master.getSubimage(j*tileWidth,
i*tileHeight, tileWidth, tileHeight);
                        }
                }

                return tiles;
        }

        /**
         * Paints the image to the window, when needed. Mostly used by
         * other classes, and not directly by a human programmer.
         * @param pen the interface to the window
         * @since 1.0
         */

        @Override
        public void paint(Graphics pen) {
                pen.drawImage(image, leftOffset, topOffset, null);
        }

        /**
         * Paint the window a solid color, on both surfaces.
         * @param color the color to paint it
         * @since 1.0
         */

        public void paintBackground(Color color) {
                Graphics pen = image.createGraphics();
                pen.setColor(color);
                pen.fillRect(0, 0, image.getWidth(), image.getHeight());

                pen = backImage.createGraphics();
                pen.setColor(color);
                pen.fillRect(0, 0, backImage.getWidth(),
backImage.getHeight());
        }

        /**
         * Pause execution for a little while.
         * @param millis number of milliseconds to pause
         * @since 2.0
         */

        public static void sleep(int millis) {
                try {
                        Thread.sleep(millis);
                } catch (InterruptedException e) {
                        // do nothing if sleep got interrupted
                }
        }

        /**
         * Translate the window-relative x coordinate into the active-area-
relative x coordinate. This might be useful
         * @return the width
         * @since 2.0
         */
```

```
	public int translateWindowX(int windowX) {
		return windowX - leftOffset;
	}

	/**
	 * Translate the window-relative x coordinate into the active-area-
relative x coordinate. This might be useful
	 * @return the width
	 * @since 2.0
	 */

	public int translateWindowY(int windowY) {
		return windowY - topOffset;
	}

	/**
	 * Update method called by the object itself, to reduce flicker.
	 * @since 2.0
	 */

	@Override
	public void update(Graphics g) {
		paint(g);
	}

	/**
	 * Outputs the window as a PNG file.
	 * @param filename the name of the file to be created
	 * @since 1.0
	 */

	public void writeAsPNG(String filename) {
		// output to disk
		try {
			ImageIO.write(image, "PNG", new File(filename));
		}
		catch (IOException ex) {
			System.err.println("ERROR: Couldn't write to file \""
+filename+"\".");
		}
	}
}
```