Zen Guide

John Lee

February 22, 2013

Contents

1 How Do I Change The Window Size?

```
public static ZenInstance create(int width, int height, String options)
```

When a Zen method is called, a window is automatically created at a default size. You can change the size it is created at by calling this method and specifying the width and height.

You can also use the `String options` and it will do a scan through the string for various (case insensitive) keywords and then enable certain options based on what it finds. The only currently implemented option is `"stretch"` which will stretch the screen to fit whatever size the user resizes the window to. If you do not wish to use this feature, you can simply set options to `null`.

Unfortunately, I don't think there is a simple way to change the size of the window or the size of the canvas that gets drawn on at run time.

2 My Screen Is Flickering, What Does Flip Buffer Do?

```
public static void flipBuffer()
```

Usually whenever you draw something to the Zen window, it will be added right away. However this will cause flickering since the window can get updated in the middle of updating the screen, causing you to see incomplete graphics.

However, when `flipBuffer()` is called for the first time, it stops Zen from updating the window immediately, and all drawing methods are done on a back buffer. This way, you can make sure that you have finished drawing before updating the screen.

When you want to update the screen, simply call `flipBuffer()` again. (If you do not, the screen won't change even when you draw more things.) I recommend putting this at the end of your drawing loop, and also right before you have any pauses to wait for input.

3 How Do I Draw Shapes?

```
public static void setColor(int red, int green, int blue)
public static void setColor(Color color)
```

These two methods allow you to either use RGB values from 0-255, or a `Color` value to change the current color objects get drawn in.

The color that objects get drawn in is determined by the last call to this method, so this is an important method to use if you don't like monochrome graphics.

If something is not being drawn to the screen, always make sure you are calling this function to make sure you aren't drawing your polygons in the same color as the background.

```
public static void drawLine(int x1, int y1, int x2, int y2)
```

Draws a line in the current color from `x1`, `y1` to `x2`, `y2`.

Pretty self explanatory. The line thickness cannot be changed directly through Zen as of right now I believe.

```
public static void fillRect(int x1, int y1, int width, int height)
```

Draws a filled in rectangle at the given X-Y Coordinates with the given `width` and `height`.

This method, other than being used to draw rectangles, is also commonly used to clear the screen.

```
public static void fillOval(int minX, int minY, int width, int height)
```

This draws an oval filled with the current color at the given X-Y coordinates with the given `width` and `height`.

The X and Y coordinates are NOT the center of the oval. Instead, imagine a rectangle being drawn with this method. The oval is then the largest ellipse that can fit inside that rectangle.

```
public static void drawArc(int x, int y, int width, int height,
int startAngle, int arcAngle)
```

This draws PART of the OUTLINE of an oval.

Unlike the `fillOval()` method, the X and Y coordinates ARE the center of the oval. The part of the outline drawn starts at the given `startAngle` (where 0 degrees is at 3 o' clock) and the "extent" of the angle is determined

by `arcAngle` (rotating counter-clockwise). Note that the angles are specified in DEGREES not RADIANS.

4 How Do I Draw Text?

```
public static void drawText(String text, int x, int y)
```

Writes the given `String` in the current color at the given X-Y coordinates on screen.

```
public static Font setFont(String fontname)
```

Changes the current font that the text gets drawn in. Just like `setColor()`, the last call to this method determines which font is used.

Additionally, if you add "-n" to the end of a the font name, it will change the size of the font to the given number. For instance "Arial-16" will let you write text in 16 pt. Arial.

```
GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontNames = ge.getAvailableFontFamilyNames();
```

is a piece of code that will create an array of strings that will give you a list of avail- able fonts on your system. I am including this because the Linux systems do not have a lot of the typical fonts.

5 How Do I Draw Images?

```
public static void drawImage(String filename, int x, int y)
public static void drawImage(String filename, int x, int y, int
width, int height)
```

These two methods will draw the image given by `filename` at the given X-Y coordinates. You can optionally resize it by giving it a `width` and a `height`. The first time you draw the image, it is loaded into memory and then cached so you don't have to worry about the image being re-loaded every time you use this method.

```
public static Image getCachedImage(String filename)
```

The image with the given filename is returned, again, you don't need to worry about it being loaded more than once with this method.

```
public static void drawImage(Image image, int x, int y)
```

`public static void drawImage(Image image, int x, int y, int width, int height)`

Draws the given image at the given X-Y coordinates optionally resized to the given `width` and `height`. Using this with `getCachedImage()` is basically the equivalent of what the other `drawImage()` methods do.

`public static Graphics2D getBufferGraphics()`

Returns a reference to the screen's graphics upon which you can directly perform draw operations on. You can use this method along with the `getCachedImage()` method if you feel like messing around with some of Java's more complicated `Image` controls.

6 What Else Can I Do With Images?

```
public static BufferedImage toBufferedImage(Image src, boolean alwaysCopy)
```

This converts the src `Image` you give it into a `BufferedImage`. A `BufferedImage` is just like a regular `Image`, only it has additional methods for editing the data contained in it.

Since a `BufferedImage` is just a cooler version of an `Image`, if `alwaysCopy` is set to false and the `Image` you give this method is secretly not just an ordinary `Image` but a `BufferedImage`, it will just return a casted version of the `Image` you gave it. This means that the data won't be copied to a new `BufferedImage`, so if you change something in the `BufferedImage` you receive from this method, the original `Image` will also be changed (since they are the same `Image`.)

```
public static BufferedImage getWindowScreenShot()
```

This returns a `BufferedImage` containing the current image on your game's window. Pretty cool? I think so.

```
public static void setClipboardImage(Image image)
```

I believe this copies the `Image` parameter given onto your clipboard, just as if you had done it in any other application. You can use this in tandem with `getWindowScreenShot()` to get totally radical screenshots of your game.

```
public static BufferedImage getClipboardImage()
```

This lets you use images that were copied onto the clipboard. Geez, with all these functions you could make an entire image editor! (*cough* MP 4 *cough*)

```
public static int[][] toRGBArray(Image img)
```

This lets you convert an `Image` into a 2 dimensional integer array, where each X-Y index of the array contains an integer RGB pixel value.

7 How Do I Get The Mouse Position?

```
public static int getMouseX()
public static int getMouseY()
```

These two methods return the current X and Y position of the mouse relative to the window.

8 How Do I Get Mouse Clicks?

```
public static long getMouseClickTime()
```

This method returns the last time the mouse was clicked. If you want to know when a mouse button has been pressed, you can store the last time the mouse was pressed, and check to see if that time has changed.

```
public static int getMouseClickX()
public static int getMouseClickY()
```

These two methods return the X and Y position the mouse was at, which may not necessarily be the current mouse position. This information may be useful to you.

```
public static int getMouseButtonsAndModifierKeys()
```

This is a slightly more advanced method which you can use to gain more mouse precision regarding mouse input (as well as some modifier keys, which I'll explain later.) What it will do is return a set of bits in the form of the integer (a bit mask). Specific bits get set to `1` whenever certain mouse button and/or modifier key are being pressed, and `0` when they are not.

In order to tell whether or not a certain bit is set, you have to bitwise and, `&`, the integer you receive with the appropriate mask for the button (which are given as `MouseEvent` constants.)

Example -

```
boolean isMouseButton1Pressed =
(Zen.getMouseButtonsAndModifierKeys() && MouseEvent.BUTTON1_DOWN_MASK) != 0;
```

This piece of code will set the `boolean` to `true` if the 1st mouse button (which is typically left click) is being pressed. There should be different mask values for buttons 1-3 (left, right, and middle click.) If you want to know a little bit more about how this code works, look up some stuff about bit masking and bitwise operators in general.

You can also mask for special modifier characters like the shift key, windows/meta key, alt, control, etc. using `KeyEvent` constants.

9 How Do I Get Keyboard Presses?

```
public static boolean isKeyPressed(char key)
public static boolean isVirtualKeyPressed(int keyCode)
```

These two methods both return true when they given key is being pressed. The difference is, the `isKeyPressed()` method takes in a character as a parameter, so 'a' will return true if the a key is being pressed, '^' if shift and 6 is being pressed, etc. `isVirtualKeyPressed()` on the other hand will return true if given specific key constants defined by `KeyEvent` are being pressed. These are things like `KeyEvent.VK_A`, `KeyEvent.VK_DOWN`, etc.

Generally `isKeyPressed()` is better for taking into account things like shift values for values that require you to use shift to access them, and `isVirtualKeyPressed()` is better when that doesn't matter as much, or when they key doesn't have a character associated with it (e.g. the arrow keys.)

10 What Does Edit Text Do?

```
public static void setEditText(String s)
public static String getEditText()
```

This wasn't totally obvious to me, so I'm writing an explanation for you.

Zen keeps track of a certain `String`. Whenever a key is "typed" that character is added to the end of it. "Typed" here means that a key is typed whenever it is first pressed, and if held long enough will continuously be typed until it is let go, similar to what would happen if a key was held down in a word processor.

If you press 'A', it will register an A being pressed, pause for a while, and then continue registering a bunch of A's (the frequency of this, and the length of the pause I believe are determined by system settings.)

The `getEditText()` method allows you to obtain the current value of this string. It's useful when you want to give the user text input on screen.

The `setEditText()` method allows you to change the contents of this string, usually to set it back to an empty `String`. This is not a good method for getting individual key presses with.

11 Miscellaneous Methods

```
public static String getAboutMessage()
```

```
return "Zen Graphics (version 0.13) Copyright Lawrence Angrave,
February 2010";
```

That is this entire method.

```
public static int getZenWidth()
public static int getZenHeight()
```

These return the width and height of the space you can draw in. If the user resizes the window this method will NOT return the window width or height.

```
public static int bound(int value, int min, int max)
```

A method that takes the given value and changes it if it is lower than the `min` or higher than the `max`.

It's a useful math method.

```
public static void waitForClick()
```

Pauses the thread and checks every 250 milliseconds for the next mouse click to occur. Presumably it's supposed to be used when multiple threads and multiple windows are opened, to pause them when one goes out of focus, and unpause them when one is clicked on again.

```
public static boolean isRunning()
```

Checks to see whether the current thread is running.

```
public static void closeWindow()
```

Closes the Zen window. When you close a Java program all the resources get released anyway, so this is probably for threaded programming.

```
public static void sleep(int milliseconds)
```

This pauses the current thread for the given number of milliseconds. Also probably used for threaded programming.