EZ, a Java Graphics Library for Introductory Programming Students

Master of Science in Computer Science

Dylan Kobayashi

Advisor: Jason Leigh

University of Hawaiʻi at Mānoa
Abstract

Across the United States colleges and universities are experiencing an increased student interest toward the computer science degree. This is probably due in part to the increased proliferation of electronic devices as well as the reported increase in jobs demanding a computer science based degree [NSF 2014]. The United States Department of Labor also predicts that the computer science field will be among the top ten with the highest job growth over the next decade[DOL 2015]. Commonly seen by many as a technology job with good future prospects there are also many students who enter the field despite not knowing much about it or result of being pressured by family. Of course, there is also a number of students entering the major who like video games thinking that is what they want to do for a job[Harris 2011]. However the computer science major is difficult for many due to a scrupulous requirement of logic and self study. Even the most simple of programs has proven complex and difficult for complete beginners. Worst is when students do the bare minimum just to get by. Failure to self study in order to truly learn the concepts will hamper a student’s prospects.

To counteract this, the student must become involved in their work. Not just involved, but prideful, willing, and happy to show their work to others. The EZ graphics library was created as an experiment to see if graphics could be incorporated into the ICS111 course. The theory was that it would help students learn to program by literally allowing them to see the effects their code has on a program. The design of interaction was structured to help reinforce object oriented concepts. Made known to the students was that all complete submissions would be shown to the rest of the class. We found that students were successfully able to use the EZ library to create programs which they understood. In fact many of the students would actively seek guidance on how to implement their vision of completion. This actually led to quickly finding the limitations of the library. Best were those, who despite the limitations, sought alternatives to produce their anticipated work.
Table of Contents

Abstract
Table of Contents
Introduction and Motivation
Related Work
  Java’s awt and swing libraries
  Java Taskforce
  Piccolo2D
  Simple Java Graphics
  Qt
  LWJGL
  JME3
The EZ library
  Features of EZ
  Comparison of HelloWorld
  Implementing a visually dynamic program
Components of EZ
  The EZ class
  EZCircle
  EZRectangle
  EZText
  EZImage
  EZLine
  EZPolygon
  EZGroup
  EZSound
  EZInteraction
Miscellaneous Mechanics of EZ
  Startup
  Painting
Evaluation
  Work produced by students
    Highlights of Assignment 1
    Highlights of Assignment 2
    Highlights of Assignment 3
  Future work and conclusion
References
Appendix
  Comparison between available libraries
  History Log
  Java Docs
Introduction and Motivation

For the University of Hawaii at Manoa the computer science introductory courses are limited to the console for program interaction. Although it is the most simplistic and very easy for a learner to use, it is not impressive or visually appealing to many. In this modern age in where computers are commonplace and the apps are graphical, conveying the difficulty and effort spend is a struggle with a console based program. Those without a programming background may not understand or appreciate what the writer has spent their time on, most will not be able to look past the fact the program is just text. But it is the standard for introductory Java courses to use the console or stay within an integrated development environment editor. Even if the student knew how to make a jar file, most programs could not be executed outside of the console. Since Java also needs at minimum the JRE to execute the program, there is an additional perceived limitation or restriction.

Development of the EZ library started as a goal to get a working graphics program within one ICS111 lab period (~75 minutes). A brief deviation from the norm that was the command line. The lab would provide a brief graphical component to present possibilities of something they could build in their free time. Pong was a great target to shoot for due to its simplicity and easy identification. Although most students may not know of pong the computer game, most did know about tennis and table tennis(ping pong). The initial measure of success was getting a circle which would act as the ball to bounce off the borders of the window and two rectangles acting as paddles. The instructions had to be simple enough to follow so that the students wouldn’t become lost or overwhelmed by the code necessary to implement. Any student who completed the program should also be able to reflect upon the actions taken and identify how the different lines of code affect the visuals. Should a student be unable to explain at least briefly what they did to accomplish their result would imply that they didn’t get anything meaningful from the exercise. Thankfully many of the students did receive the exercise positively and stated they got something out of it even if they weren’t confident they could replicate it outside of lab. The biggest impression on me was made when a couple students asked if it was ok to show their results to friends.

The driving force behind creating EZ slowly shifted to the desire of helping introductory students create a program that they would be willing and proud to show someone who did not have a programming background. In the context of a university introductory course, the student should be acquiring programming skills. How does someone show they can program? While each course has homework designed to reinforce the current topics, most students discard it shortly after completion. Homework and class exercises are not commonly viewed as a worthwhile means to provide a benchmark of their ability. It wouldn’t be mentioned on their resume or advertised on their webpage as a gauge of their skill.

Implementing a graphics based curriculum is a great means to attract learners and keep their interest[ Leutenegger & Edgington 2007 and Papastergiou 2007 ]. But while graphics can help enhance a learning experience, the real benefit is showing others. For many the idea of a programmer is one who creates software as seen on computers, phones,
and other mobile devices. Creating a graphical program isn't just for the sake of making something pretty. A graphical program makes sense to the average person who uses technology. Arguably it is an expectation. It is a wonderful event when a student is able to show someone without a computer science background their work and impress them. A practical reason is being able to start building their resume and showcasing their work. Uploads to Youtube is free. Starting a portfolio of their capabilities early gives them an advantageous start over others, especially if the work they show is understandable by all. As a showcase item, when a program is displayed on a website it is often difficult to describe with text alone. This is especially true as more complexities are added to a program like user interaction. A console based program could have descriptions, but to truly understand it one would have to download and run the program. In the case of graphical programs, a recording can be made and posted. This can quickly convey the program in a clear overview and requires little investment on the observer’s part.

After one of the early versions finally became quick to use, EZ went through a number of developmental iterations with the design of usage to stress Java learning through object oriented concepts. Written in Java, it is intended for the beginner programmer (no prior experience) as a means to enhance and encourage learning through manipulation of 2 dimensional graphics. The user does not need to hassle with installing or associating files. EZ is supplied as a self contained source file(.java) without package associations that works with a standard distribution of the Java Development Kit(JDK) version 8. The user need only add a copy into the working project folder. Interaction with EZ is done through static context, and eliminates the need to worry about correctly using Java’s awt and swing libraries. A working graphical window is achieved with one line of code. The following sections will describe work that has done before, what they did, the differences and features of EZ, followed by work produced by it, and an evaluation.
Related Work

There are a number of options available to a Java programmer with which to implement graphics with. But not as many options when it comes to graphics oriented towards a student with no background in programming. This section will go over seven different libraries that are free of cost and available for download: the native swing and awt libraries, Java Task Force, Piccolo2D, Simple Java Graphics, Qt, LWJGL, and JME3. Not all of them were designed for beginners or are limited to Java. Each was reviewed and taken into consideration during the writing and designing of EZ.

Java’s awt and swing libraries

<table>
<thead>
<tr>
<th></th>
<th>Automatically update visuals</th>
<th>Supports 3D Graphics</th>
<th>Supports reactive event driven updates</th>
<th>Designed for usage beyond simple</th>
<th>Drag and Drop supported</th>
<th>Class Platform compatibility without native OS files</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene graph concepts</th>
<th>Object oriented visual element manipulation</th>
<th>Allows for a dynamic program that changes over time</th>
<th>Received feature update this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes*1(1)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>AWT and SWING</td>
<td>Yes*2(2)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes*3(3)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No*4(4)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen().
2. User must account for changes through the paintComponent() method overload.
3. Awt and swing libraries come with the JDK.
4. Canvas contents are not object oriented in manipulation. However if the container is a JComponent or JFrame, there is some limited transform class methods.

The JDK includes the awt and swing libraries, but since they need each other to function correctly it would be more accurate to describe them as two parts of the integrated graphics components. While one might wonder, “why not use the libraries (awt and swing) that the JDK comes with?” the amount of knowledge needed to use the libraries does not permit someone without a programming background to use them. A clear understanding of programming syntax is necessary for usage. In addition one must also be comfortable with object creation, manipulation, referencing, events, and polymorphism. It is possible to setup a template file ready for a user to add graphics statements, but this would only present an environment where only static images could be easily implemented with little room for further complexity. To create a dynamic program the user would have to become familiar with the way awt and swing works. This includes the structure and hierarchy of implementing and interacting with the JFrame, JPanel, and JComponent.

Although complex, the swing system is great for creating a graphical user interface(GUI), which is what it was designed for. But when it comes to drawing shapes and using image files it becomes cumbersome. Getting graphic related elements on screen requires usage of the canvas. Using the canvas is done through the user creating a primitive awt shape and then painting it on. But once painted it remains until the canvas is cleared. Manipulating the awt object further will not change the displayed shape. It must be painted again. To create a moving shape, the user must have swing setup with a canvas area, then
correctly keep track of time and use awt shapes to stamp a design onto the canvas, while clearing it between each movement. The bare minimum of producing a window with an editable canvas creates an additional learning burden which can detract (or even deter) a student’s learning, especially a beginner. The Oracle website also offers the java3d library as a separate downloadable package that can be installed and integrated with JDK. But it has similar initial requirements just to get to a state where one can start adding 3d elements. In addition, the user will need to understand how to integrate it with the JDK compiler and work with 3d attributes. The usage is similar to the canvas system that awt and swing used. Which is rather unusual with consideration that most modern 3d libraries are based around the scene graph. [Oracle 2015]

Java Taskforce

<table>
<thead>
<tr>
<th></th>
<th>Automatically updates visuals</th>
<th>Supports 3D Graphics</th>
<th>Supports native event driven updates</th>
<th>Designed for usage beyond learning</th>
<th>Drag and Drop Install</th>
<th>Cross-Platform compatibility without native 3D files</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene graph concepts</th>
<th>Object oriented visual element manipulation</th>
<th>Allows for a dynamic program that changes over time</th>
<th>Received feature updates this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes <em>(1)</em></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Java Task Force</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes <em>(2)</em></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

2. The Java Task Force by default is available as a jar file that must be associated. There is an optional source code zip which can be used in a drag and drop manner.

The Java Taskforce was written by Eric Roberts of Sanford University. It is a collection of packages that were condensed into one jar for graphics implementations comprised of classes and custom wrappers for quickly interfacing with Java’s built in swing and awt libraries. The files are provided as a jar or source files in package hierarchy. Integration using the jar file can be difficult for someone without experience, as no instructions were given. Since the source files are available in a zip, it would be quicker to download that instead, then drag and drop into the project folder. The last updated was made in 2008 and the packages they used were based off those available at the time for JRE 6. With quick startup in mind, this was a vast improvement over regular usage of swing and awt. Designed with the goal to make things “easier to teach Java to first-year computing students without having those students overwhelmed by its complexity.“ Their tutorials and teaching material usually involved one concept and used one of their classes to help illustrate and provide context. Each class represented a different type of window that would show up and had a run method used to fill the contents of the window.

Included were three different program subclasses: ConsoleProgram, DialogProgram, and GraphicsProgram which created a window to act as an alternative to the console, JPanel dialog prompt, and JPanel canvas. The ConsoleProgram was just a manipulatable window that acted as a console. It was the DialogProgram and GraphicsProgram classes which streamlined the setup process and removed the hassle of working with swing and awt. Instead of needing to correctly create and setup JFrames, JPanels, and JComponents one could use the DialogProgram and/or GraphicsProgram which had a run method that manipulated the contents of the window. There was no need to hassle learning how to setup swing.
case of the GraphicsProgram, extending the class and create a run method to immediately start creating awt shapes to be drawn on the canvas. The DialogProgram had a lot of custom code designed to allow one line statements control of the full dialog window. Later tutorials then encouraged the learner to read through the Program class which the previous three are descendents of. The goal is to learn the differences and how they interacted with the JFrame and JPanel to eventually get the user to directly work with swing classes.

Quick and powerful for a beginner, yet it was very limiting due to the intention of eventually weaning the user from this library in order to directly use swing. In place of the main method became the run method. The run method activated and ran only once. For the DialogProgram this presented a number of limitations mainly because the contents of the window were determined by the statements made in run. The entirety of the program had to be fully accounted for and entirely written in run. This was possible because the dialog presented always had at most one button and looked for one specific data type of input. It was not designed to look or act like an interface beyond command line input replacement. The GraphicsProgram suffers from a similar effect of the run method. Anything that would be shown in the window had to be created within the run method. While it was possible to create moving graphics it was entirely reactive to the user’s mouse input. There was no updater loop that could be hooked into or overridden to animate shapes. However as mentioned before, this was designed to create a starting point for a user to learn about the awt and swing library with the intention that learners eventually stopped using the training classes. [Roberts 2008]

### Piccolo2D

<table>
<thead>
<tr>
<th>Feature</th>
<th>EZ</th>
<th>Piccolo2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically updates visual</td>
<td>Yes(1)</td>
<td>No</td>
</tr>
<tr>
<td>Supports 3D graphics</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Supports reactive event driven updates</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Designed for usage beyond learning</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Drag and Drop Install</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Cross platform compatibility without native IDE files</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Does not require prior programming knowledge</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Utilizes object oriented concepts</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Object oriented visual element manipulation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Allows for dynamic program that changes over time</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Received updates this year</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen()
2. Piccolo2D has an animator method which will make updates based upon user provided values. Otherwise reactive changes will happen automatically.

Piccolo2D is written by Ben Bederson from the University of Maryland. Piccolo2D still receives bug fixes, but has not had a major updates since early 2013. There used to be a set of direct downloads available to get the necessary files, however currently the website states that the only way to get a copy of Piccolo2D is through the Maven Central repository. This is not beginner friendly, but as of this writing the websites implies that an easier means of getting a copy will be available after they finish switching their code migration from their current host. Designed as a “Zoomable User Interface” that supports animation(movement of objects), Piccolo2D content editing is done through a scene graph. The library itself is built upon Java’s swing library. A Piccolo2D object is added to a JFrame starting with the PCanvas, but beyond this JPanels do not need to be handled. After creating a JFrame, further manipulation is through the Piccolo2D classes. In actuality the PCanvas is actually an extended JComponent.
While they called the class PCanvas, it acts much more like a scene graph. The difference between a canvas and a scene graph is that a canvas is drawn onto. Changes made to a canvas are like a real canvas. If you draw on it, it will keep the drawn shape and no change. Any additions and changes will be done in addition to the previous brushes. When a scene graph has things added to it, the resulting view is not static. A scene graph has objects added to it and will change what it displays based upon the properties of the objects being tracked. In this way the scene graph is dynamic and will alter its view whenever a change is made to any of the objects that it told to observe. Whereas the canvas will alter what it displays only when told based upon the properties of the given object at that time.

The PCanvas by itself is just a blank window. By adding nodes, the window can be populated with visual elements. It comes with some standard elements like text and image. The text can be customized with location, size, and font to name a few. The image class extends the PNode which is the parent class that all nodes should extend from. Being the parent class, all visuals are descendents of PNode which contains a paint method. It is this paint method that will render onto the window and actually acts most like a canvas. Java awt shapes can be made and painted within the Node’s width and height. While an awt shape can have coordinate values that extend beyond the PNode, anything outside of the bounds will not be rendered.

Piccolo2D has a number of interesting ideas. The first able to natively pan and zoom through the canvas and all nodes added to it. The second is being able to specify an animation action. Piccolo2D accomplishes this by having a render thread and providing methods to interact with it. First is that an animation action object must be created. There are three choices for: location, scale, and rotation. Each has a starting and end value. Then a node must be specified and given a starting time. Finally that same node must have an end time applied to it. Finally, are the mouse events. Each node has an add input event listener method. Any node can have a listener added to it that must correspond to a premade event class that Piccolo2D is setup to listen for. Another is the idea that each of the nodes can have a tag added to it which is used for node searching.

Piccolo2d suffers similar issues as the Java Task Force library. Although it still received bug fixes, according to the website it was actually designed for Java 2. Many of the features that Piccolo2D implemented to improve upon swing have had their impact lessened by Java’s updates to swing. It still handles a couple things automatically like zoom and pan. But beyond this all other features can be done natively in swing with a relatively similar amount of effort aside from the behind the scenes setup done in PCanvas. The animation methods are rather bulky and an equivalent could be done by the user through an updater method that tracks time. The latest implementations of swing frames, panels and components can implement one of the Listener interfaces and achieve the equivalent of the addEventListener methods without the hassle of object creation. [Bederson 2015]
Simple Java Graphics

Simple Java Graphics is written by Cay Horstmann for an introductory programming course he teaches at Udacity. He has also written the Java Concepts textbook that the ICS111 course at University of Hawaii at Manoa used previously. The library is distributed as source files in a zip. Most interesting is how lightweight the library is. There are only five classes that can be instantiated, which is limiting in options, but the implementation is very simplistic. The classes build upon and use the JDK included awt and swing library. To alleviate the learning curve, window creation and setup is done automatically. A user only needs to create one of the implemented elements and activate the draw method. The draw method makes a static call to the Canvas which checks if there is an active window. If a window already exists, then the object reference is added to a list of tracked elements. Any time an element’s property is changed the window will update. Once created, an object can be moved, scaled or have it’s color changed.

Unlike the previous libraries, the major negatives of this library is the desire for more features rather than the glaring issues with implementation. The dimensions of an object must be made on creation. Although possible to move and scale, the height and width ratio must be maintained after creation. As a simple example, this prevents the library from creating an horizontally growing loadbar unless the user premakes multiple objects based off the minimum, or starts off screen. A nice property to have that is not implemented is the ability for objects to rotate. Once an object is added, they cannot be removed. It is possible to “remove” an object by placing it off screen, but it still occupies space in memory. The placement offscreen must always be towards the third quadrant (negative x and negative y). An object is added to the canvas by activating the draw method. But the canvas size cannot be specified by the user. This is because the canvas will automatically adjust the window borders based upon the elements being tracked. Less burden on the user, but creates a situation where the window may change throughout program run time. If a user wanted the window to remain a specific size, this may present challenges. The final major issue with implementation is the naming of classes. Of the classes part of the library (including the support classes), five share names with an already existing JDK class: Canvas, Color, Line, Rectangle, and Shape. This has the potential to create confusion when searching for documentation online. Especially if a user begins to use classes outside this library. Naming conflicts aren’t that bad in themselves when a beginner isn’t likely to use the counterparts, but to some extent this reduces the capability of the library by not using the more robust classes that are naturally included with the JDK. Should a learner progress sufficiently to begin exploring the library, they will be
faced with the fact that it uses both its own custom classes and explicit awt references for the classes that share a name. Overall this is a good example of a quick and simple library that a beginner can use. While the implementation has some negative aspects, they are few and more than anything this library presents a good implementation that mainly lacks features. [Horstmann 2013]

Qt

<table>
<thead>
<tr>
<th>Supports 3D Graphics</th>
<th>Supports event driven updates</th>
<th>Designed for usage beyond learning</th>
<th>Drag and Drop Install</th>
<th>Cross Platform compatibility without native OS files</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene graph concepts</th>
<th>Object oriented visual element manipulation</th>
<th>Allows for dynamic program that changes over time</th>
<th>Received feature updates this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes*1)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Qt</td>
<td>Yes</td>
<td>Yes*2)</td>
<td>Yes</td>
<td>Yes*2)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen().

2. Qt supports 3d through 3rd party plugins.

Qt is made by Haavard Nord and Eirik Chambe-Eng from the Norwegian Institute of Technology in Trondheim. Originally written as a C++ library, a recent version called Qt Jambi provides Java bindings to the C++ library. Qt is actively updated as a commercial product and is mainly used to create reactive GUI's. Very fast and versatile in it's own right, the design goals weren't quite aligned with 2d graphics portrayal and manipulation. However Qt has been around since 1991 and many of the documents gave informative descriptions of their design practices and why. Once document explained the scene graph, the term had been encountered before, but prior to reading this document the full conceptual meaning was unknown. Many of the described scene graph implementations have been considered for EZ. [Trolltech 2015]

LWJGL

<table>
<thead>
<tr>
<th>Supports 3D Graphics</th>
<th>Supports event driven updates</th>
<th>Designed for usage beyond learning</th>
<th>Drag and Drop Install</th>
<th>Cross Platform compatibility without native OS files</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene graph concepts</th>
<th>Object oriented visual element manipulation</th>
<th>Allows for dynamic program that changes over time</th>
<th>Received feature updates this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes*1)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LWJGL</td>
<td>Yes</td>
<td>Yes*3)</td>
<td>Yes</td>
<td>Yes*3)</td>
<td>No*2)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen().

2. LWJGL does not support the scene graph for 2d graphics.

3. LWJGL operates in a canvas manner for 2d graphics.

The Lightweight Java Game Library(LWJGL) was sought out for its presence as a commercially proven library. It is still being updated and has released version 3 just this year. Used in games like Minecraft and Revenge of the Titans, the intention was to find something used in industry and evaluate the mechanics that a user is presented to work with. Minecraft is a fairly well known game. Wouldn't someone give more weight to a library used in something they were familiar with? Quite simply the setup process alone is enough to scare a
beginner away. Being a library that contains bindings for OpenGL and OpenAL to name a few, it is built upon an already complex base. But how did the bindings allow a user to generate graphics, play audio, or detect input?

As complex as it was, the options given to a user were diverse and meaningful, despite that at times the correct usage was roundabout or confusing. The methods aligned with the design goals outlined in many of the Qt documents. Given an object, the base manipulations were translate, scale and rotate. But the means of manipulation were by passing the objects to handler methods. The reason for this was that a cube object was actually just a container of four different vertices. A shape was just a collection of points that specified their connection order. The library sought to give the user as many options as possible and let them choose how to manipulate it, but done so by passing an object through handlers rather than placing the manipulators on the object itself. This is most likely the result of interfacing with OpenGL. The ironic part is perhaps that the agreement in design with Qt was restricted to 3d manipulations based upon the scene graph. LWJGL’s 2d implementation was canvas based like the previously mentioned libraries. The audio was simplified to creating one object given a valid file path. At this point that object became the primary means of interacting with the audio file. Play, pause, and stop were the primary modes of interaction with various other features like skip to and get current position. This was much different from Java’s standard implementation which needed at least four statements to do little more than play a valid audio file. Also significantly different was the means of interaction. The previous libraries were all modeled after Java’s event handler to detect input. A listener had to be defined in an entirely new class with an appropriate method to handle events that were generated through input. In LWJGL’s case, there was one static input class that could be asked if a specific key was down. All interaction detection was done through that one class as queries to the state of input devices rather than creating event handlers. In addition the option was given to pass an object that could act a listener. While LWJGL isn’t recommended for a beginner, the design concepts provided a different approach that could reduce overall complexity. Terminology was also a major consideration since it would be best to start acclimating a beginner to the vocabulary used within industry. [LWJGL Team 2015]
**JME3**

<table>
<thead>
<tr>
<th></th>
<th>Automatically updates visuals</th>
<th>Supports 3D Graphical</th>
<th>Supports receive event driven updates</th>
<th>Designed for usage beyond learning</th>
<th>Drag and Drop install</th>
<th>Cross platform compatibility without native OS files</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene graph concepts</th>
<th>Allows for a dynamic program that changes over time</th>
<th>Object oriented visual component manipulation</th>
<th>Allows for a dynamic feature update this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes*1)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JME3</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen().

Review of JMonkeyEngine3(jme3) was done with a similar expectation of LWJGL. Find an implementation used by industry to discover features that could be used to reduce complexity while allowing the learner to gain transferable skills and become accustomed to the jargon to be encountered. The 3d aspects of jme3 are actually built upon LWJGL, which is used as the renderer. Jme3 is notable as one of the few 3d Java game engines available which is still receiving updates. Two of its competitors are based on jme3. Ardor3D is a fork of jme3 [Slack 2014] while Env3D was built atop jme2 [Madar 2011]. As expected, setup and integration with a project is beyond the capabilities of a beginner. Jme3 was also based around the scene graph. Add objects, and based upon their properties, the view into the world would alter. The major difference was that unlike LWJGL, all 3d objects were fully defined with methods relating to what could and could not be done to it[JME 2015]. This was a significant improvement in usability and was more in line with object oriented programming.
The EZ library

Features of EZ

EZ has a couple of goals it tries to achieve with regards to usage experience. One of the major goals is to allow the user a quick and simple means of generating graphics on the screen. This is achieved through encapsulating away the interaction with Java’s awt and swing packages. The primary intention behind this is to reduce the barrier of entry for usage. While learning awt and swing allows the user to create complex GUI systems natively, they are not beginner friendly topics. Awt and swing required the user to work with object instantiation, object references, events, observers, and polymorphism to name a few.

Java is an object oriented program. The second goal is to help reinforce the idea of object oriented programming through manipulation of the visual elements. But with swing the majority of Java tutorials teach graphics concepts through one panel that acts as a canvas. However the canvas approach relies upon the user to properly maintain the drawing area. Drawing onto the canvas is akin to creating a stamp, altering the value to get the correct design, then applying [Oracle 2015]. EZ uses the scene graph approach for handling graphics. The user can choose from a variety of primitive shapes like circles, rectangles, and lines. But once an object has been added it remains in the scene and only the variables need to be altered. Changing what is seen becomes a matter of identifying and altering the corresponding object through assigning different properties or method calls.

A third goal of EZ is to create a piece of software with similar aspects to industry implementations such that when the move is made onto something better they can at least transfer some, if not most, of the skills learned. As much as possible EZ has used classes that are part of the JDK(like awt.Color), used naming schemes that adhere to industry terminology, and has implemented features that would be common in a commercial product for graphics manipulation.

EZ takes a lot of inspiration from the effortless approach that Simple Java Graphics had displayed. The goal was to emulate the ease of use of Simple Java Graphics and combine it with the power and versatility of the scene graph. Naming schemes were enforced to acclimate students to the common idioms and jargon they would encounter as a programmer. The features implemented were selected to provide a versatile and powerful framework that were also implemented by competitive products like Qt, LWJGL, and jme3.
<table>
<thead>
<tr>
<th>Library</th>
<th>Automatically updates visual</th>
<th>Supports 3D graphics</th>
<th>Supports receive event driven updates</th>
<th>Designed for use beyond learning</th>
<th>Drag and Drop install</th>
<th>Cross-platform compatibility without native OS ties</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene Graph concepts</th>
<th>Object oriented visual component manipulation</th>
<th>Allows for dynamic program changes over time</th>
<th>Received feature update this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes*1</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AWT and SWING</td>
<td>Yes*2</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No*4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Java Task Force</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Piccolo2D</td>
<td>Yes*6</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Simple Java Graphics</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Qt</td>
<td>Yes*8</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LWJGL</td>
<td>Yes*9</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No*10</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JME3</td>
<td>Yes*10</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen().
2. User must account for changes through the paintComponent() method overload.
3. Awt and swing libraries come with the JDK.
4. Canvas contents are not object oriented in manipulation. However if the container is a JComponent or JFrame, there is some limited transform class methods.
5. The Java Task Force by default is available as a Jar file that must be associated. There is an optional source code zip which can be used in a drag and drop manner.
6. Piccolo2D has an animator method which will make updates based upon user provided values. Otherwise reactive changes will happen automatically.
7. Simple Java Graphics has a short section on time calculations and how to gradually apply changes over time.
8. Qt supports 3d through 3rd party plugins.
9. LWJGL does not support the scene graph for 2d graphics.
10. LWJGL operates in a canvas manner for 2d graphics.

**Comparison of HelloWorld**

The EZ library is designed to have a low learning curve such that someone without previous programming would not be intimidated by having to learn how to use EZ in addition to learning the language. With this goal in mind, one of the criterion for ease of use was HelloWorld. When learning any programming language, the traditional first program is HelloWorld. In java this is typically five lines of code (depending on what is counted) as follows:

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

A beginner does not understand the meaning or purpose behind most of the code. From the teaching context of a first lesson, main() is the starting point, braces contain the contents, and the print line causes output to the console. Students are encouraged to change inside the parenthesis of the print, but the rest should be copied exactly and their meaning will be explained in time, since confusion may ensue if explained in depth. Creating a graphical HelloWorld should be reasonably similar in ease of use, otherwise the library’s learning curve will become an additional mental burden. There is only so much code that “just has to be accepted for now” before learning turns into memorization.
In order to have an equivalent HelloWorld using EZ, the following steps must be taken:
First get a copy of the EZ.java file.
Put it into the working project directory.
Create the HelloWorld.java file:

```java
import java.awt.Color;
public class HelloWorld {
    public static void main(String[] args) {
        EZ.initialize(100,100);
        EZ.addText(50, 50, "Hello World", Color.black);
    }
}
```

Two more lines than before, the most jarring being the Color import. Explained as a means to let the computer understand what color is. The two lines in main are significantly more complex than the standard print, but allow for broader variety of tinkering. EZ.initialize() is the statement which creates the window. The given parameters are for width and height respectively. EZ.addText(), does as the name implies, adds the text to the window. The first two parameters represent the coordinate to center the text around, followed by the text to display, and finally the color. The tinkering aspect are to encourage the learner to change the parameter values, restart the program, and observe the effects. The possible changes and effects that can be made to the program strictly through value changes are comparatively large. Using this as a sandbox, the user gets immediate feedback. The feedback might not make sense at first, but through the visuals they can literally see the effects of the changes made to the code.

After the window is setup(EZ.initialize), users can now add various elements to the screen by using the add commands of EZ. There are the standard elements like the aforementioned text as well as circle, rectangle, and line. There are three additional elements that can be added: image, sound, polygon,and group which are more complex, but do add a significant degree of versatility to the programs that can be created. Each of the add methods returns a reference to the object made, allowing the user to store it in a variable and make changes at a later time. A more indepth review of the different elements will be made in the Components of EZ section.

**Implementing a visually dynamic program**

Getting graphics on the screen is fairly simple and straightforward when it comes to static generation. However, for dynamic graphics, a loop is necessary. EZ will only update the screen when an object is added. However after an object is added, any attribute changes will not be visible, since the screen is not being updated. The swing, JTF, and Piccolo2D were
able to create a graphically dynamic program through the design of being reactive to input from the user. For example, if a button is pushed, all changes necessary are made and done so immediately. Piccolo2D had an interesting way to get around this by making an animation class. But the problem was that implementation was very bulky and non-intuitive. One had to make an animation object. Pass it the starting state of the elements which needed to move. Then alter the properties of the element to where it should end up and pass it again. Then give a start and end time for the animation. The time was based upon program run time. Finally the animation had to be set to active [Bederson 2015]. SJG would immediately adjust the visuals whenever a change was made. Their tutorials go over how to track time and leave it as something the user should implement [Horstmann 2013]. Jme3 being a game engine, had update functions on the object which were called at specific time intervals [jME 2015].

Each of these are complex and difficult for a beginner. Instead, a different approach was taken. EZ has a refreshScreen() method. Anytime that method is called, all changes to the objects being tracked will be taken into account and the visuals on screen will adjust accordingly. The most important aspect to this is that anytime the method is called, if it was called within 0.0166_ seconds of the previous call, it will wait until 0.0166_ seconds have passed before updating the screen. The wait will halt all following statements as well. That number was specifically chosen to emulate 60 frames per second(fps). The user will no longer need to keep track of time, so long as they comply with the 60 fps. Although 60 fps is one of the industry standards for update speeds, a beginner may not be accustomed to this which is why the option to change the delay is available by calling EZ.setFrameRate() [LWJGL and jME 2015]. That method takes a parameter to represent how many frame updates should occur within one second and adjusts the delay appropriately.

The following program builds upon Hello World. This time the text will move across the screen toward the right. Once it has fully pass off the right side of the screen it will appear back on the left side, as though it were wrapping around.

```java
import java.awt.Color;
public class HelloWorld {
    public static void main(String[] args) {
        EZ.initialize(100, 100);
        EZText t = EZ.addText(50, 50, "Hello World", Color.black);
        while(true) {
            EZ.refreshScreen();
            t.translateBy(1, 0);
            if (t.getXCenter() - t.getWidth()/2 > 100) {
                t.translateBy(-1 * (100 + t.getWidth()), 0);
            } //end if
        } //end while
    } //end main
} //end class
```
In the above code, a novice may find a couple lines to be rather jarring. A standard sight for persisting program, while(true) causes much confusion on first introduction. Especially when there is no exit condition. But it is this loop which makes the animation possible. The next line is necessary to view the changes. Without EZ.refreshScreen(), the view will never update. Requiring this call to update the screen was deemed an acceptable obligation for one major reason, timing. An earlier version of EZ provided a constantly updating view whenever any element’s property was changed. But in actuality this had to be done by continuously telling the window to redraw in a different thread. This meant that everything was being constantly redrawn as quickly as possible. One problem was that the updates were too fast. It was possible to control the update speed by calculating the change in time and altering by that amount of change. However, this proved incredibly difficult for someone who never dealt with tracking time before. But code was still done in a loop, and the calculation was usually placed at the bottom or top. Using a thread also came with an increased amount of problems since the values between threads were not always immediately available. Thus the decision was made where if the program was to be dynamic it was necessary for the user to tell the screen to refresh. By default the refresh will attempt to maintain a constant 60fps.

Components of EZ
The EZ class

```
static void initialize() This will setup EZ for usage.
static void initialize(int width, int height) This will setup EZ for usage.
static void refreshScreen() Used to repaint the application.
static int getWindowHeight() Used to get the window height not including the frames.
static int getWindowWidth() Used to get the window width not including the frames.
static int getDeltaTime() Used to get the difference in time since the last refresh.
static int getCurrentFrameRate() Will return the current frame rate that EZ is updating at.
static void setFrameRate(int fr) Sets the frame rate, which controls how fast the program will attempt to update itself.
static void setFrameRateASAP(boolean b) Calling this method will tell EZ whether or not it should be updating As Soon As Possible (ASAP).
static boolean isFrameRateASAP() Returns whether or not the program will update ASAP.
boolean pushBackOneLayer(EZElement ve) Will push the given element back one drawing layer.
boolean pushToBack(EZElement ve) Will push the given element to the back of the drawing layer.
boolean pullForwardOneLayer(EZElement ve) Will pull the given element to the front of the drawing layer.
boolean pullToFront(EZElement ve) Will pull the given element to the front of the drawing layer.
static EZElement getTopElementContainingPoint(int x, int y) Returns the topmost element that contains the point.
static ArrayList getAllElementsContainingPoint(int x, int y) Collects and returns all elements containing the specified point.
static EZCircle addCircle(int x, int y, int w, int h, java.awt.Color c, boolean filled) Adds a circle to the window.
static EZRectangle addRectangle(int x, int y, int w, int h, java.awt.Color c, boolean filled) Adds a rectangle to the window.
static EZText addText(int x, int y, java.lang.String msg, java.awt.Color c) Adds text to the window.
static EZText addText(int x, int y, java.lang.String msg, java.awt.Color c, int fs) Adds text to the window.
static EZImage addImage(java.lang.String filename, int x, int y, java.lang.String msg, java.awt.Color c, int fs) Adds text to the window.
static EZLine addLine(int x1, int y1, int x2, int y2, java.awt.Color c) Adds a line to the window.
static EZLine addLine(int x1, int y1, int x2, int y2, java.awt.Color c, int thickness) Adds a line to the window.
static EZPolygon addPolygon(int[] x, int[] y, java.awt.Color c, boolean filled) Adds a polygon to the window.
static EZGroup addGroup() Adds a group to the window.
static EZSound addSound(java.lang.String file) Adds a sound to the window.
```

The EZ library currently has eight different elements that can be created and an additional three support classes. First is the EZ class itself. It is through the initialize() method that EZ becomes active. Calling initialize() will set up and show the window as well as readying EZ for element creation. Apart from the add methods which is necessary for use, the next most commonly used method may be refreshScreen() which was just covered enables dynamic visuals. There are some standard utility methods like getWindowWidth(), getWindowHeight(), getDeltaTime(), get and set frame rate methods. These are fairly standard when compared to other libraries. Most people choose to manually specify the window width and height, which might make the get width and height methods redundant, however the overload option for initialize() takes no parameters and automatically tries to make the window as large as possible. While the default of refreshScreen() is to emulate 60fps, the actual difference in time between updates may vary. This is where getDeltaTime() is useful to know exactly how much time has passed. Rather than precalculating and assuming the amount of time which has passed between refreshes, getDeltaTime() will give the exact amount as tracked by EZ. This also reduces the difficulty of implementing time counters.
The last set of methods available in EZ are for element layering and detection. EZ graphics are 2d and are drawn on the canvas system of swing. If two elements are created at the same position, the one on top will always be the one created last. An element which visually appears behind the second is considered to be on a lower layer. The opposite is true where an element that appears in front of another is considered to be on a higher layer. By nature of the canvas system, it is not possible for two elements to be on the same layer, if at the same coordinate, one will always appear in front of the other [Oracle 2015]. But if at some point it is desired that an elements layer should change, there are four methods available: pushToBack(), pushBackOneLayer(), pushToFront(), and pullForwardOneLayer(). They do exactly as their names describe. If it is not possible for an element to move (only element or is already backmost / topmost element), then the method doesn’t do anything. Because all elements must be on separate layers, when moving an element between layers, if necessary, other elements may have their layers changed to accommodate the move. The last set of methods available in the EZ class are for detecting elements at a point. Given a valid coordinate on the window, getTopElementContainingPoint() or getAllElementsContainingPoint() can be used with it. The top element refers to the element with the highest draw layer value, the one visibly on top of others. If the method to get all elements at the specified point is used, then an ArrayList will be returned with the start having the lowest layer valued elements and the end having the highest layer valued elements.
The EZElement class is actually an abstract class from which all visual elements of EZ inherit from. This was designed as a means to standardize the properties and capabilities of all visual elements. Since a visual element has to be displayed on screen, there are some basic methods that are consistent throughout all children. They have a show() and hide() method which will affect whether or not they should be drawn, but also affects the method which get element at point in the EZ class. All elements have a standard set of properties that affect how it is drawn: coordinates, width, height, rotation, and scale. In addition to the show() hide() there are also some properties that do not affect all elements, but still are part of them: fill status and color status. The fill mainly applies to the circle, rectangle, and polygon. All other elements cannot have their fill status changed, and querying it will return true. Color can
be applied to the circle, rectangle, polygon, text, and line. Like the fill status, all other elements cannot have fill status changed, and querying it will return black.

The standard properties are related to coordinate location. Origin(0,0) is actually located at top left corner of the window. Although most math classes place origin at the center (or bottom left when only showing positive values) the decision was made to go with the industry standard of origin in the top left [W3C 2011, LWJGL and jME 2015]. When the coordinates are set and retrieved from the object, it is always with respect to the center placement. This a common implementation of commercial products and becomes especially meaningful when manipulating something that is not uniform in shape or appearance. Retrieval of coordinates is done through the methods getXCenter() and getYCenter(). However the center values are not assigned by using a method like set x or y center. Instead the center can be altered through translateBy(), translateTo(), and moveForward(). Again, terminology from commercial products and standards when manipulating a three dimensional object [W3C 2011, Apple Inc 2012, Trolltech, LWJGL, and jME 2015]. When using translateBy() it must be given two parameters representing how many pixels to shift from its current location. When using translateTo() the object’s center will be placed at the given point. The last one, moveForward() with move forward a pixel amount specified by the parameter. But the direction of forward is actually based upon the rotation of the object. All objects start with 0 rotation, making forward toward the right of the screen. The unusual part of this method is that an object cannot be located at pixel 1.5 or 3.39. Afteral, a pixel is the smallest unit of display on a screen. The decimal places must still be tracked though. Consider the case where a 45 degree angle is applied to an element. If it was told to moveForward(1), meaning by one pixel in the direction it is facing, the activation will not move the object. Should the value be truncated, at 45 degrees the object would never move regardless of how many times it was told to move forward by 1 pixel. The retrieval of the object’s center location will always be an int value due to pixel coordinates. But the actual values may have decimals that are discreetly being tracked.

The manipulation of an object’s rotation can be done through four different methods. The most concise being rotateTo(), which will set the rotation to the given value. When using rotateBy(), it will be an additive adjustment to the current rotation. However through feedback it was found that not everyone was comfortable with using rotateBy() due to not always being fully confident which way the element would rotate. As result turnLeft() and turnRight() were added. The effect of turnRight() is the exact same as rotateBy(), but the naming makes it very clear which way the object will turn. Using turnLeft() will subtract the given value from the current rotation rather than being additive.

When scale manipulation was added, the implementation was done to mimic the mechanics of available engines. The original size is a scale of 1. Twice the original size would be 2. Half the size of the original would be 0.5. A scale of -1 would retain the same size, but invert the visuals. There is the standard way of manipulating scale, using scaleTo() which sets the scale to the given value. Unlike translate and rotation, when using scaleBy() it will perform a multiplicative action. The reasoning behind it has to do with the value representation. If the original size is 1, then double is 2. The double of that is 4. When telling an objec scaleBy(2), this can be translated as, double the size of the object, which isn’t additive, but multiplying by
2. Anytime an object should double in size, the scale should be multiplied by 2. If the object should be tripled, scaleBy(3), then multiple by three. This also applies to reduction. If an object should be halved, then pass 0.5. [W3C, LWJGL, and jME]

The identity() function will reset an object to position 0,0 with its normal size (scale of one), and reset the rotation back to zero. There are three methods that deal with parenting: setParent(), getParent(), and removeParent(). Each of these will be explained in more detail under EZGroup. Finally are the isPointInElement() and layer manipulation methods: pushToBack(), pushBackOneLayer(), pullToFront(), and pullForwardOneLayer(). The isPointInElement() returns true if the given coordinate is within the bounds of this object. This is will return true even if the object is not showing. The layer manipulation methods have the exact same effect as those in EZ with the same name, except these do not take a parameter since the element being affected is the one the method is called from.

**EZCircle**

```
EZElement

EZCircle

<table>
<thead>
<tr>
<th>void setHeight(int h)</th>
<th>The circle can have its height changed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setWidth(int w)</td>
<td>The circle can have its width changed.</td>
</tr>
</tbody>
</table>
```

Creating an EZCircle is done with EZ.addCircle(). It must be given parameters for center x and y, width and height, color, and whether or not it should be filled. Using the addCircle() method will also return a reference to the object to allow for modification later in the program. EZCircle is a descendent from EZELEMENT and has all the previously mentioned methods. On significant mechanic to note is that it is possible for the circle to have a width and height that are not equal.

**EZRectangle**

```
EZElement

EZRectangle

<table>
<thead>
<tr>
<th>void setHeight(int h)</th>
<th>The rectangle can have its height changed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setWidth(int w)</td>
<td>The rectangle can have its width changed.</td>
</tr>
</tbody>
</table>
```

A basic element much like the EZCircle, it has the core functionality as outlined in EZELEMENT. Creating one is done with EZ.addRectangle() and must be given parameters for center x and y, width and height, color, and whether or not it should be filled. And just like the
circle, it is possible for the rectangle to have a width and height that are not equal. For both the circle and rectangle, changing the width or height will not affect the center. Rather it will expand out an appropriate amount from the center location. Very important is what happens when an odd value is given. The center coordinate, width, and height forms the basis of the shape generated as well as the bounds. When a shape has an odd value for width and height, it will retain the value. The element will retain its correct size, but depending on the pixel coordinate the drawn center may actually be off by one pixel depending on the row or column it is drawn on. This is result of the fact that a pixel is the smallest unit on a monitor.

**EZText**

<table>
<thead>
<tr>
<th>EZElement</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EZText</th>
</tr>
</thead>
</table>

| String | getFont() Will return the name of the font currently being used. |
| void   | setFont(java.lang.String name) Will attempt to set the font to the specified parameter. |
| int    | getFontSize() Returns the size of the font. |
| void   | setFontSize(int f) Changes the size of the font. |
| String | getMsg() This will return what text is currently being displayed |
| void   | setMsg(java.lang.String m) Will set the displayed text to the given parameter. |
| static void | printAvailableFontsToConsole() Will print all available fonts to console. |
| static String[] | getAllFontNames() Will return a String array containing all names of fonts available for usage specific to this machine. |

This is actually the first class that might be introduced, with respect to the Hello World program. The EZ.addText() has three overloads available. The standard one was used in the Hello World example and takes four parameters: x and y center coordinates, the message to display, and the color to draw the text in. The second way adds a parameters to the previous call. The added parameter represents the font size to use, by default the font size is 10. Finally is adding one more parameter representing the name of the font to use. By default the font style is whatever the standard value for the JVM on the machine is set to. If it is unknown what fonts are available there are two static methods available in EZText. The first is printAvailableFontsToConsole() and the second is getAllFontNames(). The easiest to use would be the first one which does exactly as named, prints all available fonts. It can also be used without any EZText objects created. The negative is that usually this involves running the program just to see what fonts are available and they taking note before actually created the intended program. The getAllFontNames() method is much more complex as it will return a String array containing all available font names.

EZText inherits from EZElement and has all associated method. But the width and height of a text object cannot be manually specified. Rather the width and height are the result of the text displayed, font type, and font size. The fonts available are those available to the system made in the true type font.ttf format. As result even when using the same font size, but different font type, the width and height are almost always different. After a text object has
been created it is possible to get and set the displayed text using getMsg() and setMsg(). The font type and size can also be retrieved and changed with getFont(), setFont(), getFontSize(), and setFontSize() respectively. In addition, while possible to change the color of the text, it is not possible to change the fill status. Text is always filled. Like the circle and rectangle, the center coordinates will not change unless specified. Altering the message will still retain the center coordinate. While most people are familiar with text editors where adding or reducing text always affects the right side, the center anchor is compliant with commercial products. Should a user wish to expand or reduce from the right, they can offset based on half the width. If the user desires to use a font that is not natively available within the system, it is also possible to specify a ttf file.

**EZImage**

<table>
<thead>
<tr>
<th>boolean</th>
<th>hasFocus() Returns whether or not the image has a focus area set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>setFocus(int xTopLeftCorner, int yTopLeftCorner, int xBottomRightCorner, int yBottomRightCorner) Will set a focus area on the image that will be displayed instead of the entire image.</td>
</tr>
<tr>
<td>void</td>
<td>releaseFocus() If a focus area has been set, it will be released and the entire image will be shown.</td>
</tr>
</tbody>
</table>

Adding an image is done by EZ.addImage(), with three parameters for file name and coordinates(x,y) for the center. The filename can also specify a valid path. Should the specified file or path be invalid, EZ will not crash. Instead the image will be replaced with a placeholder rectangle stating that the image could not be loaded. EZImage is able to load the different variations of jpeg, png, bmp, and gif. However in the case of the gif, only the first frame will be rendered. Alpha channel will be preserved allowing transparency. However, unlike text, once an image has been specified, it cannot be changed.

While the core methods of EZElement are available, the image prevents manipulation of a couple things. Width and height are derivatives of the image itself, they cannot be manually specified. The nature of an image also prevents the object from specifying a color. Any query made will return black by default. All images are filled.

In order to reduce memory overhead and increase efficiency, the EZImage class has some background properties which keeps track of all loaded images. Should a user create another image object, but using the same image file as a previous object, it will not reload the image file into memory again. Instead, all EZImage objects that refer to the same image file will contain references to the memory allocation. This allows a user to potentially created as many EZImage objects as desired without depleting memory at a rate of file size per object.

By default an image object will show the entire image. However this isn't always desired. In such a case, setFocus() can be used. The setFocus() method takes four parameters, the first two representing the top left corner of a focus area on the image, and the second two representing the bottom right corner. Should a focus area be set, only the portion
of the image within the specified rectangle will be shown. Like the screen, origin is top left of the image. However, it is also possible to specify a coordinate that is not contained within the image, as well as give the corners in opposite order. When specifying a coordinate that is not contained within the image, the non-image area will be displayed as fully transparent. Giving the corners in opposite order will invert the image. In addition to changing what is displayed, setFocus() will alter properties of the object. The specified area will be moved and adjusted such that the center of the focus will be placed at the specified center coordinates of the object. In addition, width and height will correspond to the focus area. This is particularly important when the focus area contains more than the image itself. Even though extending beyond the image will render as a transparency; the center placement, width, and height will be based off of the full zone.

**EZLine**

![EZLine Diagram](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getThickness()</td>
<td>Returns the current thickness of the line.</td>
</tr>
<tr>
<td>setThickness(int t)</td>
<td>Increases the thickness of the line to given parameter.</td>
</tr>
<tr>
<td>getX1()</td>
<td>Returns the x coordinate of point 1.</td>
</tr>
<tr>
<td>getX2()</td>
<td>Returns the x coordinate of point 2.</td>
</tr>
<tr>
<td>setPoint1(int x, int y)</td>
<td>Changes the coordinate of point 1, which will probably result in a change of center and rotation.</td>
</tr>
<tr>
<td>newY1()</td>
<td>Returns the y coordinate of point 1.</td>
</tr>
<tr>
<td>newY2()</td>
<td>Returns the y coordinate of point 2.</td>
</tr>
<tr>
<td>setPoint2(int x, int y)</td>
<td>Changes the position of point 2, which will probably result in a change of length, center and rotation.</td>
</tr>
</tbody>
</table>

When creating a line with EZ.addLine(), four values are used to specify two points with which to connect to form the line. Color must also be passed as a parameter. The overloaded method adds a sixth parameter to specify a line thickness. Thickness default to 1 if not provided, and cannot go below 1. Should a negative or 0 value be provided, EZ will reset to 1. Unlike the other elements, a center coordinate is not specified, instead the center coordinate is derived from the two points given. Also unlike the other elements, rotation does not always default to zero. It is derived from the slope between the first and second point, which makes it possible to start with a rotation greater than 90 degrees if the first point has coordinate values greater than the second point.

While the center and rotation is a derivative of the two provided points the methods for translation, scale, and rotation still work as normal. Thickness is affected by scale. In addition to the standard methods, the line also provides ways to interact with the individual points. Should the points be directly set, the center and rotation will be recalculated based off the new coordinates of the altered point.
EZPolygon

The last EZ object that is visually perceivable is the EZPolygon. This class does not have any unique methods, but does require setup before initialization. When using EZ.addPolygon(), it must be given two int arrays, color and fill status. The two arrays contain either all the x values or all the y values, as result their cardinality must match in order to have a pair of values for each point. The order the values are given will specify the line which forms the perimeter of the polygon. The last point will be connected to the first as a means of closing the polygon. In this manner the enclosed area will be used as the polygon and the center calculated based on the rectangle properties needed to enclose the entire polygon. Due to this, it is possible to specify coordinates that place the center outside of the polygon’s actual bounds. While complex for the user to instantiate, the intention is to allow for shapes beyond the standard circle, rectangle, and line.

After a polygon has been constructed the translate, scale, and rotate functions will work as normal. The width and height cannot be altered as they are a derivative of the points specified. Color and fill status can be set like the circle and rectangle. Although the initial center coordinate is based upon the bounding rectangle, when using the point in element methods, they will be applied to the polygon’s actual bounds rather than the bounding box.

EZGroup

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean addElement(EZElement e)</td>
<td>Adds an element to the group.</td>
</tr>
<tr>
<td>boolean removeElement(EZElement e)</td>
<td>Will attempt to remove the specified element.</td>
</tr>
<tr>
<td>ArrayList getChildren()</td>
<td>Returns an ArrayList of EZElements containing all children of this element.</td>
</tr>
</tbody>
</table>

Each of the above visual elements describe can be placed within a group. Creating a group, EZ.addGroup(), doesn’t take any parameters. By default the group starts at origin. It is not possible to see a group, rather the group is meant as a means for multiple objects to be manipulated as one. By adding an object to a group, the object will have its center coordinates adjusted so that visually it will not have moved at all, but now its coordinates are with respect to the center of the group being the origin[W3C 2011 and jME 2015]. There are two ways an object can be added to a group. EZElement has a setParent() method that all visual elements have implemented. Call the method from the specified element and pass the group which should become a parent. The other way is through the group, call the addElement() and pass the object to become a child. Only groups can be parents. However a group can be the child of another group. An element is only allowed one parent. Trying to set a parent when one already exists, will not do anything. First parent-child association must be removed, then parent can be applied to a different pair. When a child is removed from a
parent, visually it will not appear to have moved, but all corresponding properties will be adjusted to keep the current visual positioning.

A group cannot have color or fill specified, as the group acts as a container and has no visible traits. The width and height are calculated through a bounding box over all children, which may need to be applied recursively for groups containing groups. The real power of a group comes from manipulating all associated elements as one. When the group is translated, all children move with it. The same applies to scale and rotate. But, rotate is applied as one element. When a group rotates, like the other elements, rotation will be done around the center. Should a child be off center, when rotating a group, that child will rotate around the group’s center rather than it’s own.

EZSound

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void play()</td>
<td>This will play the sound file from wherever the current position is.</td>
</tr>
<tr>
<td>void pause()</td>
<td>Will pause the sound at it's current position.</td>
</tr>
<tr>
<td>void stop()</td>
<td>This will stop the sound and reset the position back to the start.</td>
</tr>
<tr>
<td>void loop()</td>
<td>Will play from the start and loop the sound…</td>
</tr>
<tr>
<td>int getFrameLength()</td>
<td>Returns how many frames are held within this sound file.</td>
</tr>
<tr>
<td>int getFramePosition()</td>
<td>Returns the current frame of the sound file.</td>
</tr>
<tr>
<td>void setFramePosition(int pos)</td>
<td>Sets the position in frames from which to continue playing.</td>
</tr>
<tr>
<td>long getMicroSecondLength()</td>
<td>Returns the total length of the sound file in microseconds.</td>
</tr>
<tr>
<td>long getMicroSecondPosition()</td>
<td>Returns the current position in microseconds.</td>
</tr>
<tr>
<td>void setMicroSecondPosition(int pos)</td>
<td>Sets the position in microseconds from which to continue playing.</td>
</tr>
</tbody>
</table>

The sound is not visible and does not inherit from EZElement, as result is not actually tracked by the standard handler within EZ. Creating a sound with EZ.addSound() needs to only be passed the sound file which could be a path. Currently EZ is only able to play .wav files. Once the object has been created there are the standard controls of play(), pause(), stop(), and loop(). Using the play() method will prompt the audio. The full clip will run and stop at the end. Using pause() before reaching the end will allow the next call of play() to resume where it left off. Otherwise using play() after it has fully completed will start again from the beginning. Using stop() instead of pause() will always reset to the beginning. Using loop() will have the same interaction that play() has with pause() and stop(), but upon reaching the end and cutting the audio feed, it will restart from the beginning.

In addition to the basic methods are: getFrameLength(), getFramePosition(), getMicroSecondLength(), getMicroSecondPosition(), setFramePosition(), and setMicroSecondPosition(). A frame in an audio file is the smallest unit of sound that is uniform in duration with one specific audio tone. Depending on the quality of audio encoding, a frame may not always be the same duration. Getting the frame length will return how many frames are within the audio clip. Getting the microsecond length will return the duration of the clip in microseconds. Getting the position will return the current value of the corresponding property
queried. Using the set position will allow the next play() call to start from the specified position [Oracle 2015].

Creating multiple sound objects that refer to the same file will actually allocate that file’s size in memory each time. This is different from EZImage, due to the difference in how audio is played. The Java AudioInputStream can only play from one location. By allocating memory for each object, if multiple objects refer to the same file they would all be allowed to play at the same time even from different points in the audio sample. This unfortunately can lead to a very large consumption of memory in a short amount of time if not handled correctly. When updating visuals, refreshScreen() must be called. However, the visual fps doesn’t always match the audio frame rate. Due to this, all audio is played in a different thread. The user doesn’t have to worry about maintaining or tracking those threads, but throughout implementation and testing this has caused issues on some occasions.

EZInteraction

<table>
<thead>
<tr>
<th>EZInteraction</th>
<th>isKeyKeyDown(char c) Used for actively checking if a key is down (being pressed).</th>
</tr>
</thead>
<tbody>
<tr>
<td>static boolean</td>
<td></td>
</tr>
<tr>
<td>static boolean</td>
<td>wasKeyPressed(char c) Checks if a character was just pressed.</td>
</tr>
<tr>
<td>static boolean</td>
<td>wasKeyReleased(char c) Checks if a character was just released.</td>
</tr>
<tr>
<td>static int</td>
<td>getXMouse() Returns the x coordinate of the mouse if it is over the window.</td>
</tr>
<tr>
<td>static int</td>
<td>getYMouse() Returns the y coordinate of the mouse if it is over the window.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseLeftButtonDown() Used to detect if the left button is down.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseLeftButtonDownPressed() Used to detect if the left button was pressed.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseLeftButtonReleased() Used to detect if the left button was released.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseRightButtonDown() Used to detect if the right button is down.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseRightButtonDownPressed() Used to detect if the right button was pressed.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseRightButtonReleased() Used to detect if the right button was released.</td>
</tr>
</tbody>
</table>

Finally is EZInteraction, which cannot be instantiated by a user. It is a support class that is automatically setup with initialization(). While there are a lot of publicly available methods, most of them are listener methods that wait for input events from the window. There are three methods which can be used to get keyboard state. The isKeyKeyDown() method takes a char as a parameters and will return true or false if the key is in the down state. The wasKeyPressed() and wasKeyReleased() methods also take a char as a parameter to check. But these methods are time based with respect to refreshScreen(). For example, starting from a neutral state a key is not down, has not been pressed, or released. When the key is pushed down logically isKeyKeyDown() will return true and wasKeyPressed() will return true. Currently the key is down so it wasKeyReleased() will return false. On the next refreshScreen(), the key is still being held down; isKeyKeyDown() will return true, but wasKeyPressed() will return false. This is because the key went down last update, not this update. The wasKeyReleased() will remain false. When the key is released isKeyKeyDown() will return false as will wasKeyPressed(). But on the update when the change happens wasKeyReleased() will return true. And on the next update, wasKeyReleased() will then return false, because the change did not happen on this update [LWJGL 2015].
To poll mouse state one is always able to get the coordinates through getXMouset() and getYMouset(). Should the cursor not be over the window, then the values return will always be -1. The window doesn’t know how far away the cursor is, only that the cursor has departed. Currently in EZ one is only able to check the state of the left or right mouse button. Like the key checks they are isMouseLeftButtonDowmn(), wasMouseLeftButtonPressed(), wasMouseLeftButtonReleased(), isMouseRightButtonDowmn(), wasMouseRightButtonPressed(), and wasMouseRightButtonReleased(). They work in the same manner as the related key checks, but rather than being able to specify a mouse button to check they are permanently linked to the left and right mouse button. Originally the mouse button had to be specified like the key checks. But this lead to a couple problems. Either use a string or an int. String wasn’t possible because there was no uniform naming scheme beyond middle button. Using an int wasn’t clear which button was being manipulated. Although it was possible to create static int variables, there was again the problem of naming scheme beyond middle button. Initial users expressed frustration with needing to specify a mouse button and after changing to the methods to be fixed to the left or right, this cleared up frustrations. In fact, so far no complaints have been voiced about being restricted to left and right.

Miscellaneous Mechanics of EZ

Startup

EZ was built upon Java’s awt and swing libraries. As mentioned before in the Hello World example, creation of a window is one method call, EZ.initialize(). The EZ class actually inherits from JPanel. As a static call, initialize will create one JFrame and instantiate one EZ object, setting the width and height to the given initialize() parameters. The EZ object is set to occupy the entire content area of the JFrame. The later static calls are made possible through the constructor storing a reference to itself in a static class variable. The constructor also initializes timing variables, creates an instance of EZInteraction, and binds the input event handlers. When binding the input event handlers, this is in reference to the EZInteraction object which implements both KeyListener and MouseInputListener. Construction of EZInteraction will also set a reference of itself to a static class variable. Methods required by the interfaces are implemented such that when they are called, the input values are tracked by modifying class variables.

Painting

The EZ object inherits from JPanel, is placed and sized within the JFrame to occupy the entire space. The visuals are done through the overloaded paintComponent() method. Normally, a JPanel would fill the contents of paintComponent() with one conceptually complete figure. The figure might include text or basic shapes. Manipulation of the figure would then normally be done through the JPanel manipulation methods. This is mainly due to
the fact that each conceptually complete figure should be in separate JPanels. This however is slightly contradictory as the JPanels themselves are incapable of actions like rotation or scale. If the content they display should be rotated, the paintComponent() override must correctly apply the adjustment [Oracle 2015]. For this reason, the implementation of EZ is that there is only ever one JPanel, all of the EZElements share the same canvas space.

The EZElement is a custom build class which defines how it appears through the paint() method. However, this method is only called by EZ, not the user. For each element that a user creates, EZ will add it to a static ArrayList. Only the elements within the static ArrayList will be drawn, which is why the EZ.add methods must be used instead of just instantiating a new object of the desired type. The consideration was made to automatically add any EZElement to the ArrayList upon construction, but was decided against with the argument of reinforcing objectification of EZ. Whenever the JPanel is repainted, usually through refreshScreen(), the ArrayList will be checked and each element’s paint function will be called. Upon call, the canvas reference is passed and the individual elements are then responsible for correctly painting themselves upon the canvas. The effect of layers is produced simply by the nature of the painting process starting from the beginning of the ArrayList and going through to the end. Altering of the layer is actually changing an object’s index location within the ArrayList. The exception to this is EZGroup objects. Children of groups are actually removed from the ArrayList and tracked by the individual group objects. Upon the group’s paint() method being called, it will then call each of its children’s paint() methods.

In order for the shape elements(Circle, rectangle, line, and polygon) to be visible, the paint method will create a corresponding awt shape and adjust the values to match the translation, scale, and rotation of the object. Initially the shape will be sized over the screen origin. Application of the translation, scale, and rotation is done through an AffineTransform object. Without AffineTransform, the shape’s vertices would have to be manually set to the correct position. Through trial and error it was found that using one AffineTransform would actually queue up the actions represented by one method call (translate, scale, rotate), implementing a stack of commands. When applied to a shape, the last command given would be applied first and progress towards the first given command. Also to be taken into account was whether or not an element was part of a group. If it was, then each of its ancestor’s transformations would have to be given before the element’s.

When it came to text and images, the difficulty was that AffineTransform could not be applied to them. In the case of text, before it could be drawn, the font had to be set. A JPanel’s canvas has to be told(via String) text to have drawn on it. The canvas itself has one font which it draws all text [Oracle 2015]. Before each EZText object is drawn, it has to change the JPanel’s font setting. Thankfully it is a canvas, so once text has been drawn, changing the font will not retroactively change all previous text draws. Initially, this also caused a problem when trying to determine an EZText object’s width and height. Should there be two text objects that specified different font type and size, the JPanel’s font setting had to be changed before each width and height query. If the font settings was not changed, the width and height calculation would most likely be incorrect because it was calculated under a
different text object's properties. This unfortunately means that polling for a text object's width and height is significantly slower than polling for any other object's.

When it comes to drawing text and images, it is the canvas itself that must be transformed. This was initially very strange, since most Oracle pages state that transformation should be applied to the content as much as possible. Conceptually it didn't match either, as transforming the canvas seemed to imply that all visuals would be affected. This was not the case. Transforming a canvas would only affect subsequent draws. When text and images had any transformation applied to them, it was the canvas which needed to be temporarily transformed. After completing the draw for that object, all transformations would then be removed from the canvas so that other object would appear correctly [Oracle 2015]. The exception to this was if an image could not be loaded. In such a case, the image could not be drawn and instead a placeholder rectangle and message (“Failed to load image”) was displayed.
Evaluation

Work produced by students

EZ was used in the assignments for ICS111 at University of Hawaii at Manoa during the 2014 fall semester. The first assignment they were given was to make a clone of Pong. The second assignment was to create a file parser. The file parser would read transform commands (translate, scale, rotate) from a file and apply them to a visual. The last assignment was to create a music video as a group effort and incorporate the file parser from the previous assignment.

Students were given roughly one month to create a Pong clone. Pong consists of one ball and two paddles on opposite ends of the screen. Each paddle is controlled by a separate player where the goal is to strike the ball into the opposing player’s goal zone. The ball should bounce off the paddle and as result can be used to block the opposing player’s attempt to score. The assignment instructions included an almost step by step process to create the ball, paddles, and controls. They were allowed the option to further enhance the game with features they found meaningful. Should a student be unsure about what to add, there were descriptions of concepts that could be implemented. Described concepts included: using images rather than the basic rectangle; speeding up the ball over time; applying spin to the ball after getting hit by the paddle; tracking score; and using a circle instead of rectangle for the ball. At least 36 students attempted for more than the base assignment requirements and uploaded videos of their result to youtube.

The requirements of assignment two was proof of creating a file reader which could detect commands and correctly apply them to a visual element. Most importantly, each command has a duration so the effects of the command were to be gradually applied over the full duration rather than happening instantly. Grading of submissions was expected to be a matter of double checking their mechanics. However over 25 students not only implemented the file parser showing it could manipulate a graphical element, they made short skits, recorded the results, and sent the video as part of their submission.

Those who had created a short skit for assignment two had an advantage, as that was the goal of project 3. Using all they had learned, choreograph a music video. While it was designed to test their programming knowledge, one major predicted difficulty would be working with people. While it was expected that all students have the technical capability, it is an entirely different matter to get multiple code files working together. Meshing their interaction is not something that could be hacked away at over a weekend, they had to plan for it. All of that in addition to deciding on a topic, working together, and communicating in a productive way. The results were better than predicted. Only a couple groups outright gave up. Everyone else at least attempted and had something moving on screen. More than half of the groups created a piece of work that was beyond expectation. The best groups created a piece of work that I did not think would be possible for this class.
Highlights of Assignment 1

The following are some images from five project:
The first was skinned with flappy bird images. Speed of the ball increased with each hit and
the background scrolled as the game progressed.
https://www.youtube.com/watch?v=NsqZQO8EZdI
The second features Mii skins.
https://www.youtube.com/watch?v=79tXY6OQKG8&feature=youtu.be

Mario themed with an impressive amount of visuals and interactions implemented.
This next one had an impressive feature where shots from the ships could affect the ball. 
https://www.youtube.com/watch?v=jVNR72Cv3Nk&feature=youtu.be
The most impressive implemented powerups. Each player started with the ability to increase the ball speed, teleport the ball, and make it invisible. Should a ball get past a player, it left a crater. 
https://www.youtube.com/watch?v=FzC37_z9tmE&feature=youtu.be

Highlights of Assignment 2

Due to the technical complexity (writing a file parser) there was a low expectation for their submitted program to prove a graphic could be properly manipulated. I thought most would do a 10 second program showing that translate, rotate, and scale all worked. It was amazing how wrong that assumption was. Many students created short scripts and added music. Here are only five of those submissions. The hardest part to convey is the animations that were involved. EZ does not support videos or playing of animated image formats (gif or apng).

Super Smash Bros Brawl staged fight: 
https://www.youtube.com/watch?v=3tbx8bWFBNw&feature=youtu.be
Itsy Bitsy Spider followed by a short space odyssey:
https://www.youtube.com/watch?v=SgjEQNNBpaA
Teenage Mutant Ninja Turtles:
https://www.youtube.com/watch?v=yvNegAZgo0k&feature=youtu.be
Mock Pokemon battle:
https://www.youtube.com/watch?v=4TQtDjTu7ME

All characters appearing in this work are fictitious.
Any resemblance to real persons, living or dead,
is purely coincidental.
Emulating 3d shapes:
https://www.youtube.com/watch?v=GPCXad5ahHw&feature=youtu.be
Highlights of Assignment 3

This project was across the board unexpected good. The following are screen captures from only three projects. And even this doesn’t do them justice for the amount of animation involved.

This first project, actually even the video doesn’t do it justice. The amount of interactions and features implemented can’t be properly shown.

https://www.youtube.com/watch?v=iRsXckWokFA
For the second project, as the professor stated “When it ended, I was left wanting to see what happened next.”
https://www.youtube.com/watch?v=d6Ozb7QmwOM&feature=youtu.be
This third project was impressive in the visual details implemented both mechanically and through editing the images themselves. Some of the characters were not only moving but animated through sprite manipulation.

https://www.youtube.com/watch?v=vT3JbCcj6Jc

Future work and conclusion

EZ was designed and created as a library to introduce beginner students to graphics manipulation in Java programs. By getting students to interact with with graphics, this would hopefully provide a means to become involved in code that deepens their interest in programming and the computer science major. Designed as a learning supplement, this has the potential to provide them with experiences that will stay with them into their career. Everything considered, EZ has realized its intended purpose of getting graphics within an introductory course where students are not expected to have prior experience. I have in my records an unexpected email from a students who started the course with little to no interest in computer science but will now attempt a minor just because enjoyable graphic assignments were part of the course. That student may believe the assignments to have been a means to implement graphics, but the understanding and implementation of mechanical was by no means a simple feat. By completing each of the assignments they have created a miniature Java version of Flash. Not just writing a script that does as desired, but creating a system that is able to read the script and correctly carry out the commands as well. To complete their music video many found that object definition, instantiation, and manipulation were necessary, core concepts of object oriented programming. Although their first encounter with the
while(true) statement was confusing, they have since become accustomed to and comfortable with the idea of a persistent program structure. While some might disregard graphics as a novelty to attract learners, being able to properly implement what they want shows a good understanding and requires more than compared to output from the command line.

While quick and easy for 2d graphics, EZ is very limiting outside of the intended usage. For example, there is no way to implement 3d graphics which seems to be a very common inquiry at the end of the semester. But it isn’t just visual capabilities, there are a number of mechanical limitations that are imposed to reduce usage complexity. The first major mechanical limitation is the input. While the limitation for mouse button detection is rarely noticed (only detects left and right), keyboard limitations are quickly encountered. The method to check requires one parameter with datatype char. The implementation was done through the event handler tracking specific characters and states. Currently, only the alphabet, digits, and spacebar can be queried. None of the special characters are being tracked or keys like arrows, shift, or enter. The second limitation of input is that it must be proactively polled. Asking the user a standard instruction like “press any key” is actually very hard to detect with EZ. First reason being that not all keys are tracked, and the second reason being that the key has be actively checked in order to know if it was pressed. This makes implementing something like a chatbox rather difficult.

While designed for 2d, there are some recurring capability questions. One common question being, “does EZ play gif images?” It does actually display the first frame of a gif, but does not play the gif. Playing a gif would require accessing the file’s meta data and exchanging out frames correctly based on that data. At the moment, there is currently no intention for resolution. Another issue is the displaying of graphical elements may result in some obscuring others if common space on the window is shared. That stacking effect is result of their draw layer. Currently it can be manipulated through movement up or down by one, or going completely to the front or back. If this was done to model industry standards, what should have been done was add a property stored as int value referred to as z index, implement a method to retrieve, and a method to set[Apple Inc 2012 and jME 2015]. While shapes and input can be detect, implementing a button takes more effort than ideal. This has to do with both mechanically what it means to be a button and the effects of pushing a button. Mechanically, a button needs to detect when it is clicked. But in EZ the user has to implement the click detection. When a button is clicked, usually something happens. Again, that must be implemented by the user and they have to understand how to integrate this within their code. While I believe EZ has a lower barrier of knowledge to implement a button than standard swing usage, it isn’t always clean or straightforward to actualize.

Finally, EZ has bugs and some features are awkward to use. Over the past year many bugs have been detect and resolved; but like any juvenile program, I am certain there are still some that remain. They just might not be currently known. Or if they are known, have not been brought to my attention. Sound in particular can be awkward to use. Currently only wav files can be read, which is another thing that can be improved upon. By nature wav files are large. And while there are currently no known memory leaks in EZ, there are many ways for a student to unintentionally consume exorbitant amounts of memory.
References

Apple Inc. Coordinate Systems and Transforms.  


Harris Interactive. STEM Perceptions: Student and Parent Study.  


Leutenegger, Scott and Edgington, Jeffery.  
A Games First Approach to Teaching Introductory Programming.  


National Science Foundation. Science and Engineering indicators of 2014.  


Papastergiou, Marina. Digital Game-Based Learning in high school Computer Science education: Impact on educational effectiveness and student motivation  
http://130.216.33.163/courses/compsci747s2c/lectures/paul/GameBasedLearning_CS_Education.pdf. 2007.


http://www.w3.org/TR/SVG/coords.html. 2011
Appendix

Comparison between available libraries

<table>
<thead>
<tr>
<th></th>
<th>Automatically update visual</th>
<th>Supports 3D Graphics</th>
<th>Supports reactive event driven updates</th>
<th>Designed for usage beyond painting</th>
<th>Drag and Drop install</th>
<th>Cross platform compatibility without native OS files</th>
<th>Does not require prior programming knowledge</th>
<th>Utilizes Scene graph concepts</th>
<th>Object oriented visual element manipulation</th>
<th>Allows for a dynamic program that changes over time</th>
<th>Receiving feature updates this year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZ</td>
<td>Yes*(1)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AWT and SWING</td>
<td>Yes*(2)</td>
<td>No</td>
<td>Yes</td>
<td>Yes*(3)</td>
<td>Yes</td>
<td>No</td>
<td>No*(4)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Java Task Force</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes*(5)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Simple Java Graphics</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes*(6)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Qt</td>
<td>Yes</td>
<td>Yes*(7)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes*(8)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LWJGL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No*(9)</td>
<td>No*(10)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JME3</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No*(11)</td>
<td>No*(12)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. EZ will automatically update the view whenever a visual is created. Further manipulation must be manually performed with refreshScreen().
2. User must account for changes through the paintComponent() method overload.
3. AWT and swing libraries come with the JDK.
4. Canvas contents are not object oriented in manipulation. However if the container is a JComponent or JFrame, there is some limited transform class methods.
5. The Java Task Force by default is available as a Jar file that must be associated. There is an optional source code zip which can be used in a drag and drop manner.
6. piccolo2d has an animator method which will make updates based upon user provided values. Otherwise reactive changes will happen automatically.
7. Simple Java Graphics has a short section on time calculations and how to gradually apply changes over time.
8. Qt supports 3d though 3rd party plugins.
9. LWJGL does not support the scene graph for 2d graphics.
10. LWJGL operates in a canvas manner for 2d graphics.

History Log

V1.13
2/8/15
Documentation updates.

-----------------------------

V1.12
1/10/15
Fixed an error with sound playback.

-----------------------------

V1.11
1/9/15
Removed all marked deprecated code.
Cleaned code to remove all compile warnings.
V1.10
1/8/15
Fixes made to EZImage.
If a focus area is set getWidth and getHeight methods now return the width and height of the focus area rather than the image original width and height.
If a focus area is set isPointInElement() will return true if the point is within the new adjusted area based upon the width and height of the focus area around the center instead of using the entire image boundaries.

Changes made to EZInteraction.
Added getXMouse and getYMouse method which will return the corresponding coordinate of the mouse if it is on the window. If not, then the methods will return -1. This was made to remove the following methods.

Depreciated getXMouseMove, getYMouseMove, getXMouseClick, getYMouseClick.
The get move methods just returned the location of the mouse if it was over the window.
The click checks should be used to determine which button was clicked. Otherwise the location of the cursor at time of click is its position on the window at that time. The only difference is that the get click methods returned -1 if the click methods returned false, but the cursor is still on window.

V1.09
11/10/14
Major updates to EZImage, EZInteraction, and EZSound.

EZImage now allows for subsection focusing. Significance is ability to use sprite sheets.
public void setFocus(int xTopLeftCorner, int yTopLeftCorner, int xBottomRightCorner, int yBottomRightCorner)
The four int parameters represent two points. Upper left and lower right to form a rectangle.
The area marked by the rectangle within the image will be displayed rather than the whole image.
In this manner it is also possible to flip the image based upon the position of the two points.
public void releaseFocus()
Will release the focus and revert to displaying the entire image.
public boolean hasFocus()
Will return true if a focus area has been set, otherwise false.

EZInteraction now allows for better mouse button checks with respect to left and right mouse buttons. Much like keys, it is possible to check if the left/right mouse button was pressed, is down, or was released.
public static boolean wasMouseLeftButtonPressed()
public static boolean wasMouseRightButtonPressed()
public static boolean isMouseLeftButtonDown()
public static boolean isMouseRightButtonDown()
public static boolean wasMouseLeftButtonReleased()
public static boolean wasMouseRightButtonReleased()
The getXMouseMove() and getYMouseMove() have also been changed to detect mouse movement even if the button is down.
EZSound has bug fixes and a number of support methods added.

```java
public void play()
    Will now play based upon position on the file.
public void stop()
    Will now stop the sound file and reset the position to the start.
public void pause()
    Has been added, this will halt the sound at its current position. Using play() will resume from that position.
public int getFrameLength()
    Will return the number of frames in the sound file.
public int getFramePosition()
    Will return the current position with respect to frame number.
public long getMicroSecondLength()
    Will return the length of the sound file in microseconds.
public long getMicroSecondPosition()
    Will return the current position with respect to microseconds.
public void setFramePosition(int pos)
    Will set the current position to the specified frame number.
public void setMicrosecondPosition(int pos)
    Will set the current position to the specified microsecond position, but limited by frame association.
public double getLevel()
    Might return the amplitude of the current position. Doesn't work on all systems due to sound driver support.
```

-------------

V1.08
9/2/14

Change to the background flushing so that Windows machines do not have a right and lower ghosting zone.

-------------

V1.07
9/1/14

Implemented all font setting feature of 1.05 without the bug where the last drawn EZText element settings were used for all EZText elements.

-------------

V1.06
8/31/14

Rollback to v1.02 due to a bug where all EZText elements were using the values of the one which was drawn last.

-------------

V1.05
8/30/14
EZText has been modified to allow loading a font from a .tff file through setFont().
Added support methods and made changes to setFont() and paint().

V1.04
8/25/14
Method added to EZ:
   public static EZText addText(String fontName, int x, int y, String msg, Color c, int fs)
Allows the user to specify a font name when adding an EZText element.

V1.03
8/24/14
Added the ability to change the font of the EZText elements.
To support this additional methods have been added to EZText:
   public static void printAvailableFontsToConsole()
   public static String[] getAllFontNames()
   public voidsetFont(String name)
   public String getFontName()

   Added a variable to track the font name.
   Edits made to constructor, paint, getWidth, getHeight.

V1.02
8/20/14
Fixed a bug with EZ.refreshScreen() where it was actually getting about 50% more than the specified frame rate.
   Added a silent error tracker for situations that did not need to halt the program.

V1.01
8/19/14
Logo by Jason Leigh

V1.00
8/19/14
Released

Java Docs
Class EZ

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel
          EZ

All Implemented Interfaces:

public class EZ
  extends javax.swing.JPanel

Interaction with the EZ class should be done through the public static methods. EZ extends a JPanel which are among the few things that can be put into the JApplet. They way they work and how to interact with them is significantly different from JFrames. The standard usage of EZ will involve EZ.initialize() to create a window. Then usage of the add methods to place elements on it. The EZ.refreshScreen() method must be called in order to update the visuals. The standard update rate is 60fps. Majority of the EZ methods will not work unless EZ has been initialized.

Author:
Dylan Kobayashi

See Also:
Serialized Form

### Nested Class Summary

**Nested classes/interfaces inherited from class javax.swing.JComponent**

- javax.swing.JComponent.AccessibleJComponent

**Nested classes/interfaces inherited from class java.awt.Component**

- java.awt.Component.BaselineResizeBehavior

### Field Summary

**Fields**

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Field and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static EZ</td>
<td>app</td>
</tr>
</tbody>
</table>
|                   | Used for external referencing.

**Fields inherited from class javax.swing.JComponent**

- TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

**Fields inherited from class java.awt.Component**

- BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

**Fields inherited from interface java.awt.image.ImageObserver**

- ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

### Constructor Summary


## Constructors

### Constructor and Description

**EZ(int w, int h)**

Calling the constructor for the EZ class should never be done manually.

## Method Summary

### All Methods

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Static Methods</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static EZCircle</td>
<td>addCircle(int x, int y, int w, int h, java.awt.Color c, boolean filled)</td>
<td>Adds a circle to the window.</td>
</tr>
<tr>
<td>static boolean</td>
<td>addElement(EZElement ve)</td>
<td>Adds an element for EZ to track.</td>
</tr>
<tr>
<td>static boolean</td>
<td>addElement(EZElement ve, int index)</td>
<td>Adds an element for EZ to track at a specific draw layer.</td>
</tr>
<tr>
<td>static EZGroup</td>
<td>addGroup()</td>
<td>Adds a group to the window.</td>
</tr>
<tr>
<td>static EZImage</td>
<td>addImage(java.lang.String filename, int x, int y)</td>
<td>Adds an image to the window.</td>
</tr>
<tr>
<td>static EZLine</td>
<td>addLine(int x1, int y1, int x2, int y2, java.awt.Color c)</td>
<td>Adds a line to the window.</td>
</tr>
<tr>
<td>static EZLine</td>
<td>addLine(int x1, int y1, int x2, int y2, java.awt.Color c, int thickness)</td>
<td>Adds a line to the window.</td>
</tr>
<tr>
<td>static EZPolygon</td>
<td>addPolygon(int[] xp, int[] yp, java.awt.Color c, boolean filled)</td>
<td>Adds a polygon to the window.</td>
</tr>
<tr>
<td>static EZRectangle</td>
<td>addRectangle(int x, int y, int w, int h, java.awt.Color c, boolean filled)</td>
<td>Adds a rectangle to the window.</td>
</tr>
<tr>
<td>static ESound</td>
<td>addSound(java.lang.String file)</td>
<td>Adds a sound to the window.</td>
</tr>
<tr>
<td>static EZText</td>
<td>addText(int x, int y, java.lang.String msg, java.awt.Color c)</td>
<td>Adds text to the window.</td>
</tr>
<tr>
<td>static EZText</td>
<td>addText(int x, int y, java.lang.String msg, java.awt.Color c, int fs)</td>
<td>Adds text to the window.</td>
</tr>
<tr>
<td>static EZText</td>
<td>addText(java.lang.String fontName, int x, int y, java.lang.String msg, java.awt.Color c, int fs)</td>
<td>Adds text to the window.</td>
</tr>
<tr>
<td>static java.util.ArrayList&lt;EZElement&gt;</td>
<td>getAllElementsContainingPoint(int x, int y)</td>
<td>Collects and returns all elements containing the specified point.</td>
</tr>
<tr>
<td>static int</td>
<td>getCurrentFrameRate()</td>
<td>Will return the current frame rate that EZ is updating at.</td>
</tr>
<tr>
<td>static int</td>
<td>getDeltaTime()</td>
<td>Used to get the difference in time since the last refresh.</td>
</tr>
<tr>
<td>static EZElement</td>
<td>getTopElementContainingPoint(int x, int y)</td>
<td>Returns the topmost element that contains the point.</td>
</tr>
<tr>
<td>static int</td>
<td>getWindowHeight()</td>
<td>Used to get the window height not including the frames.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
</tbody>
</table>
| static int | getWindowWidth()  
Used to get the window width not including the frames. |
| static void | initialize()  
This will setup EZ for usage. |
| static void | initialize(int width, int height)  
This will setup EZ for usage. |
| static boolean | isElementAtPoint(int x, int y, EZElement ve)  
Given an x and y coordinate, will check if that point is within the given element. |
| static boolean | isFrameRateASAP()  
Returns whether or not the program will update ASAP. |
| static boolean | isTopElementAtPoint(int x, int y, EZElement ve)  
Will check if the given element is the top most element at the specified point. |
| void | paintComponent(java.awt.Graphics g)  
This method is in charge of painting the screen. |
| boolean | pullForwardOneLayer(EZElement ve)  
Will pull the given element to the front of the drawing layer. |
| boolean | pullToFront(EZElement ve)  
Will pull the given element to the front of the drawing layer. |
| boolean | pushBackOneLayer(EZElement ve)  
Will push the given element back one drawing layer. |
| boolean | pushToBack(EZElement ve)  
Will push the given element to the back of the drawing layer. |
| static void | recurseGroupAddingToArrayList(EZGroup group, java.util.ArrayList<EZElement> elems)  
Designed as a recursive method to collect all children and add them to the given arraylist. |
| static void | refreshScreen()  
Used to repaint the application. |
| static void | removeAllEZElements()  
Clears out all visual elements that EZ is tracking. |
| static void | removeEZElement(EZElement ve)  
Removes one specified visual element that EZ is tracking. |
| static void | setBackgroundColor(java.awt.Color c)  
This method will set the background color to the given color. |
| static void | setFrameRate(int fr)  
Sets the frame rate, which controls how fast the program will attempt to update itself. |
| static void | setFrameRateASAP(boolean b)  
Calling this method will tell EZ whether or not it should be updating As Soon As Possible(ASAP). |
| static void | trackedErrorPrint() |

Methods inherited from class javax.swing.JPanel
getAccessibleContext, getUI, getUIClassID, setUI, updateUI

Methods inherited from class javax.swing.JComponent
getListeners, getlocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVisibleChangeListeners, getVisibleRect, getWidth, getx, gety, grabFocus, hide, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInitialMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container

add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getKeyPressed, getKeyPress, getLastName, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getRootNestChangeListeners, getSize, getToolkit, getTreeLock, getFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeComponentListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, show, toObject, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

dep

public static E2 app

Used for external referencing.
Constructor Detail

EZ

public EZ(int w, 
    int h)

Calling the constructor for the EZ class should never be done manually. You should be using EZ.initialize(). Creates an instance of EZ and sets it as primary content pane and initiates values as necessary.

Parameters:
    w - value in pixels of how large to make the width of the inner content area.
    h - value in pixels of how large to make the height of the inner content area.

Method Detail

addCircle

public static EZCircle addCircle(int x, 
    int y, 
    int w, 
    int h, 
    java.awt.Color c, 
    boolean filled)

Adds a circle to the window. Returns the circle for later manipulation. If not immediately assigned to a variable, chances are you will have this element stuck on screen which cannot be removed. As result in most cases you will want to assign it to a variable. Color must be specified. Don't forget to import Color. The filled parameter will determine whether or not the element will be a solid of the given color. If it is not filled, the inner parts will be fully transparent. Example usage:
EZCircle c; c = EZ.addCircle(200, 200, 30, 10, Color.BLACK, true);

Parameters:
    x - center.
    y - center.
    w - width.
    h - height.
    c - color.

Returns:
    true will make the element a solid color. false will outline with the given color.

addElement

public static boolean addElement(EZElement ve)

Add an element for EZ to track. Generally you should not be using this. Use the more specific add methods instead.

Parameters:
    ve - The element to add.

Returns:
    true or false based on whether or not the element was successfully added.

eleme

addElement

public static boolean addElement(EZElement ve, 
    int index)

Add an element for EZ to track at a specific draw layer. Lower numbers are on lower layers. Generally you should not be using this. Use the more specific add methods instead. The given layer must be valid and within the current range of layers.

Parameters:
```java
ve - The element to add.
index - where the element should be placed.

Returns:
ture or false based on whether or not the element was successfully added.

```
int x2,
int y2,
java.awt.Color c,
int thickness)

Adds a line to the window. Returns the line for later manipulation. If not immediately assigned to a variable, chances are you will have this element stuck on screen which cannot be removed. As result in most cases you will want to assign it to a variable. The line must be created with two points. A start and end point. The line itself will then be drawn to connect the two points. Thickness less than 1 will be automatically increased to 1. Color must be specified. Don't forget to import Color. Example usage: EZLine l; l = EZ.addLine(200, 300, 600, 100, Color.BLACK);

Parameters:
x1 - The x value of point 1.
y1 - The y value of point 1.
x2 - The x value of point 2.
y2 - The y value of point 2.
c - Color to make the line.
thickness - to make the line.

Returns:
the line.

addPolygon

public static EZPolygon addPolygon(int[] xp,
int[] yp,
java.awt.Color c,
boolean filled)

Adds a polygon to the window. Returns the polygon for later manipulation. If not immediately assigned to a variable, chances are you will have this element stuck on screen which cannot be removed. As result in most cases you will want to assign it to a variable. The polygon must be created with two arrays. One holding a list of x values while the other holds a list of y value. Each index of the arrays refer to a specific point. The order of points matter, as the polygon will be drawn starting from index 0 to the end of the array. The last point will be automatically connected to the first point. Color must be specified. Don't forget to import Color. Example usage: EZPolygon p; int[] xp, yp; xp = new int[3]; yp = new int[3]; xp[0] = 100; xp[1] = 150; xp[2] = 200; yp[0] = 100; yp[1] = 200; yp[2] = 100; p = EZ.addPolygon(xp, yp, Color.BLACK, true);

Parameters:
xp - int array containing the x values for the points.
yp - int array containing the y values for the points.
c - color.

filled - true will make the element a solid color. false will outline with the given color.

Returns:
the polygon.

addRectangle

public static EZRectangle addRectangle(int x,
int y,
int w,
int h,
java.awt.Color c,
boolean filled)

Adds a rectangle to the window. Returns the rectangle for later manipulation. If not immediately assigned to a variable, chances are you will have this element stuck on screen which cannot be removed. As result in most cases you will want to assign it to a variable. Color must be specified. Don't forget to import Color. The filled parameter will determine whether or not the element will be a solid of the given color. If it is not filled, the inner parts will be fully transparent. Example usage: EZRectangle r; r = EZ.addRectangle(200, 200, 30, 10, Color.BLACK, true);

Parameters:
x - center.
y - center.
w - width.
h - height.
c - color.

filled - true will make the element a solid color. false will outline with the given color.

Returns:
the rectangle.

addSound

public static EZSound addSound(java.lang.String file)

Adds a sound to the window. Returns the sound for later manipulation. You NEED to assign this to a variable otherwise you will not be able to play the sound. Currently the sound file must be in .wav format to work. Example usage: EZSound s; s = EZ.addSound("YouGotMail.wav");

Parameters:
file - name of the sound file including extension.

Returns:
the sound.

addText

public static EZText addText(int x,
        int y,
        java.lang.String msg,
        java.awt.Color c)

Adds text to the window. Returns the text for later manipulation. If not immediately assigned to a variable, chances are you will have this element stuck on screen which cannot be removed. As result in most cases you will want to assign it to a variable. It might not be easy to calculate left or right bound until after creation since the x,y values are where the text's center will be placed. Text cannot have their width and height manually set, that will depend on the content of the text. Using this addText() method will default the text size to 10px. Color must be specified. Don't forget to import Color. Example usage: EZText t; t = EZ.addText(200, 200, Color.BLACK, true);

Parameters:
x - center.
y - center.
msg - that will be displayed.
c - color of the text

Returns:
the circle.
fs - size of the font in pixels.

**Returns:**
the circle.

### addText

```java
class EZText {  
    public static EZText addText(String fontName,  
                                  int x,  
                                  int y,  
                                  String msg,  
                                  Color c,  
                                  int fs)  
    {  
        // Implementation details
    }
}
```

Adds text to the window. Returns the text for later manipulation. If not immediately assigned to a variable, chances are you will have this element stuck on screen which cannot be removed. As result in most cases you will want to assign it to a variable. It might not be easy to calculate left or right bound until after creation since the x,y values are where the text's center will be placed. Text cannot have their width and height manually set, that will depend on the content of the text. Color must be specified. Don't forget to import Color. Example usage: EZText t = EZ.addText("Arial", 200, 200, Color.BLACK, true, 20);

**Parameters:**
- `fontName` - to display the msg in. Must be available to the system. A nonexistent font will output a console error, but will not halt the program.
- `x` - center.
- `y` - center.
- `msg` - that will be displayed.
- `c` - color of the text
- `fs` - size of the font in pixels.

**Returns:**
the circle.

### getAllElementsContainingPoint

```java
class EZElement {  
    public static java.util.ArrayList<EZElement> getAllElementsContainingPoint(int x,  
                                                                                int y)  
    {  
        // Implementation details
    }
}
```

Collects and returns all elements containing the specified point. Will not return an element which is not visible. Will not return an EZGroup. See getTopElementContainingPoint() for explanation. Polymorphism knowledge may be needed to use this method.

**Parameters:**
- `x` - coordinate of the point.
- `y` - coordinate of the point.

**Returns:**
an array containing all EZElements.

### getCurrentFrameRate

```java
class EZ {  
    public static int getCurrentFrameRate()  
    {  
        // Implementation details
    }
}
```

Will return the current frame rate that EZ is updating at.

**Returns:**
int value of the current frame rate.

### getDeltaTime

```java
class EZ {  
    public static int getDeltaTime()  
    {  
        // Implementation details
    }
}
```

Used to get the difference in time since the last refresh. Time is in milliseconds. 1 second == 1000 milliseconds. If you want to change the time between updates setFrameRate() may be what you are looking for.

**Returns:**
int value of the difference in time. Note: Standard system time counters are usually longs. However for ICS111 int is the most commonly used datatype of that family tree, and should be more than enough for delta time.

**getTopElementContainingPoint**

```java
public static EZElement getTopElementContainingPoint(int x,
    int y)
```

Returns the topmost element that contains the point. Will not return an element which is not visible. The topmost element is the one which has been drawn last making it visually appear on top others. Will not ever return an EZGroup, since technically the group itself is comprised of multiple elements. If you want the EZGroup which the given element is part of, crawl up the ancestry using getParent(). Polymorphism knowledge may be needed to use this method as it returns EZElement, not the individual subclass.

**Parameters:**
- x - coordinate of the point.
- y - coordinate of the point.

**Returns:**
- the top most EZElement that is not a group.

**getWindowHeight**

```java
public static int getWindowHeight()
```

Used to get the window height not including the frames.

**Returns:**
- int value equal to the number of pixels between the frames vertically.

**getWindowWidth**

```java
public static int getWindowWidth()
```

Used to get the window width not including the frames.

**Returns:**
- int value equal to the number of pixels between the frames horizontally.

**initialize**

```java
public static void initialize()
```

This will setup EZ for usage. Without calling this method first, none of the other EZ methods will work correctly. Window will default to use the full dimensions of the screen. Do not call this method more than once in a program run.

**initialize**

```java
public static void initialize(int width,
    int height)
```

This will setup EZ for usage. Without calling this method first, none of the other EZ methods will work correctly. Parameters will be used to determine width and height of window. Do not call this method more than once in a program run.

**Parameters:**
- width - for the content area of the window.
- height - for the content area of the window.

**isElementAtPoint**

```java
public static boolean isElementAtPoint(int x,
    int y,
    EZElement ve)
```
Given an x and y coordinate, will check if that point is within the given element. This is done with respect to world space. If the element is not showing, will always return false.

**Parameters:**
x - coordinate of the point.
y - coordinate of the point.
ve - the element to check if the point is within.

**Returns:**
true if the point is within the element. Otherwise false. Always returns false if the element is not showing.

### isFrameRateASAP

```java
public static boolean isFrameRateASAP()
```

Returns whether or not the program will update ASAP.

**Returns:**
true means it is. false means it is using the specified frame rate. 60fps is the default.

### isTopElementAtPoint

```java
public static boolean isTopElementAtPoint(int x,
                                          int y,
                                          EZElement ve)
```

Will check if the given element is the top most element at the specified point. Top most is refers to the highest draw layer meaning nothing is visually in front of it. If it is not showing, will always return false. Will not work with EZGroup.

**Parameters:**
x - coordinate of the point.
y - coordinate of the point.
ve - element to check if the point is within.

**Returns:**
true if the element is the top point. Otherwise false. Always returns false if the element is not showing.

### paintComponent

```java
public void paintComponent(java.awt.Graphics g)
```

This method is in charge of painting the screen. But do note that each of the individual elements are in charge of painting themselves. This method should not be called manually. Use EZ.refreshScreen() instead.

**Overrides:**
paintComponent in class javax.swing.JComponent

### pullForwardOneLayer

```java
public boolean pullForwardOneLayer(EZElement ve)
```

Will pull the given element to the front of the drawing layer. If the element is in a group, the element will be pulled forward once on that group's drawing layer.

**Parameters:**
ve - the element to pull.

**Returns:**
false if the element doesn't exist. Otherwise true.

### pullToFront

```java
public boolean pullToFront(EZElement ve)
```

Will pull the given element to the front of the drawing layer. If the element is in a group, the element will be pulled to the
front of that group's drawing layer.

**Parameters:**
- ve - the element to pull.

**Returns:**
- false if the element doesn't exist. Otherwise true.

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushBackOneLayer</td>
</tr>
</tbody>
</table>

**pushBackOneLayer**

```java
public boolean pushBackOneLayer(EZElement ve)
```

Will push the given element back one drawing layer. If the element is in a group, the element will be pushed back once on that group's drawing layer.

**Parameters:**
- ve - element to push back.

**Returns:**
- false if the element doesn't exist. Otherwise true.

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushToBack</td>
</tr>
</tbody>
</table>

**pushToBack**

```java
public boolean pushToBack(EZElement ve)
```

Will push the given element to the back of the drawing layer. If the Element is in a group, the element will be pushed to the back of that group's drawing layer.

**Parameters:**
- ve - element to push back.

**Returns:**
- false if the element doesn't exist. Otherwise true.

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurseGroupAddingToArrayList</td>
</tr>
</tbody>
</table>

**recurseGroupAddingToArrayList**

```java
public static void recurseGroupAddingToArrayList(EZGroup group,
    java.util.ArrayList<EZElement> elems)
```

Designed as a recursive method to collect all children and add them to the given arraylist. This will not add groups to the ArrayList, but instead will search those groups for elements adding those elements to the ArrayList.

**Parameters:**
- group - from which to start the downward search
- elems - the ArrayList to add all non-EZGroup children to.

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>refreshScreen</td>
</tr>
</tbody>
</table>

**refreshScreen**

```java
public static void refreshScreen()
```

Used to repaint the application. Without this call, any changes made to the elements will not be visibly seen. This will also control the speed of the program updates. By default it is set to 60fps. This can be modified by using setFrameRate().

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>removeAllEZEelements</td>
</tr>
</tbody>
</table>

**removeAllEZEelements**

```java
public static void removeAllEZEelements()
```

Clears out all visual elements that EZ is tracking.

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>removeEZEelement</td>
</tr>
</tbody>
</table>

**removeEZEelement**

```java
public static void removeEZEelement(EZElement ve)
```

Removes one specified visual element that EZ is tracking.

**Parameters:**
- ve - the element to remove from EZ.
**setBackgroundColor**

public static void setBackgroundColor(java.awt.Color c)

This method will set the background color to the given color. Don't forget to import the Color when using this. While standard Colors like Color.WHITE or Color.BLUE are available, it is possible to specify an rgb value using: EZ.setBackgroundColor(new Color(r, g, b)); where r,g,b are int values.

**Parameters:**
c - Color to use.

**setFrameRate**

public static void setFrameRate(int fr)

Sets the frame rate, which controls how fast the program will attempt to update itself. Note: it is very rarely possible to get an exact match of frames per second due to the time statements take to execute in addition to the fact that the division of time may not be equally distributed for that particular value of fps. Also possible to request a speed that cannot be done due to processing time of other components or hardware limitations. Will not change current fps if given value is 0 or less. A value of 1000 is the same as setting the program to update as quickly as possible, it doesn't guarantee that frame rate.

**Parameters:**
fr - an int value specifying the desired frames(updates) per second.

**setFrameRateASAP**

public static void setFrameRateASAP(boolean b)

Calling this method will tell EZ whether or not it should be updating As Soon As Possible(ASTP). Passing true will bypass the given frame rate values causing updates to occur ASAP. There will be no CPU rest, which for most programs is not necessary. Passing false will revert back to the last passed frame rate value. If none has been given, 60fps is the EZ default.

**Parameters:**
b - value true means update ASAP. false means use last specified frame rate.

**trackedErrorPrint**

public static void trackedErrorPrint()
public class EZCircle extends EZElement

The EZCircle is used to create an Ellipse type of shape. A perfect circle isn’t required and is calculated based upon given width and height. When creating a circle, the given center coordinate, width and height will specify a bounding box for the circle. From there, the circle drawn will attempt to make the most usage of the given bounding box ensuring that a line be placed along the vertical or horizontal axis the opposite sides will be symmetrical.

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description

EZCircle(int x, int y, int width, int height, java.awt.Color color, boolean filled)
Creates a circle with the given specifications.

Method Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.awt.Shape</td>
<td>getBounds()</td>
</tr>
<tr>
<td></td>
<td>This will return a Shape of the bounds of this element with respect to the world space.</td>
</tr>
<tr>
<td>java.awt.Color</td>
<td>getColor()</td>
</tr>
<tr>
<td></td>
<td>Returns the color of this element.</td>
</tr>
<tr>
<td>int</td>
<td>getHeight()</td>
</tr>
<tr>
<td></td>
<td>Returns the height of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getWidth()</td>
</tr>
<tr>
<td></td>
<td>Returns the width of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getXCenter()</td>
</tr>
<tr>
<td></td>
<td>Returns the x center of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getYCenter()</td>
</tr>
<tr>
<td></td>
<td>Returns the y center of this element with respect to local space.</td>
</tr>
<tr>
<td>void</td>
<td>hide()</td>
</tr>
<tr>
<td></td>
<td>This will prevent the element from being painted.</td>
</tr>
<tr>
<td>void</td>
<td>identity()</td>
</tr>
<tr>
<td></td>
<td>Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.</td>
</tr>
<tr>
<td>boolean</td>
<td>isFilled()</td>
</tr>
<tr>
<td></td>
<td>Will return whether or not this element is set to be filled.</td>
</tr>
<tr>
<td>void</td>
<td>paint(java.awt.Graphics2D g2)</td>
</tr>
<tr>
<td></td>
<td>The paint method controls how the element draws itself on the screen.</td>
</tr>
<tr>
<td>void</td>
<td>setColor(java.awt.Color c)</td>
</tr>
<tr>
<td></td>
<td>Sets the color of this element.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void setFilled(boolean f)</td>
<td>Will set the filled status of this element.</td>
</tr>
<tr>
<td>void setHeight(int h)</td>
<td>The circle can have its height changed.</td>
</tr>
<tr>
<td>void setWidth(int w)</td>
<td>The circle can have its width changed.</td>
</tr>
<tr>
<td>void show()</td>
<td>This will ensure the element is painted.</td>
</tr>
<tr>
<td>void translateBy(double x, double y)</td>
<td>Moves the center of the element by given x and y coordinate.</td>
</tr>
<tr>
<td>void translateTo(double x, double y)</td>
<td>Sets the center of the element to given x and y coordinate.</td>
</tr>
</tbody>
</table>

**Methods inherited from class EZElement**

boundHelper, getParent, getRotation, getScale, getWorldHeight, getWorldWidth, getWorldXCenter, getWorldYCenter, hasParent, isPointInElement, isShowing, moveForward, pullForwardOneLayer, pullToFront, pushBackOneLayer, pushToBack, removeParent, rotateBy, rotateTo, scaleBy, scaleTo, setParent, transformHelper, turnLeft, turnRight

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Constructor Detail**

**EZCircle**

```java
public EZCircle(int x,
                int y,
                int width,
                int height,
                java.awt.Color color,
                boolean filled)
```

Creates a circle with the given specifications. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addCircle() method which will perform additional background actions to get the circle to display on the window properly.

**Parameters:**
- x - center coordinate.
- y - center coordinate.
- width - of the circle.
- height - of the circle.
- color - to use when drawing.
- filled - status of whether the drawn circle should be a solid of the given color.

**Method Detail**

**getBounds**

```java
public java.awt.Shape getBounds()
```

**Description copied from class: EZElement**

This will return a Shape of the bounds of this element with respect to the world space. This is not a bounding box. This is the shape itself after transformations have been applied.

**Specified by:**
getBounds in class EZElement

Returns:
Shape instance of the bounds for this Element.

getColor

public java.awt.Color getColor()

Description copied from class: EZElement
Returns the color of this element.

Specified by:
ggetColor in class EZElement

Returns:
color of this element.

getHeight

public int getHeight()

Description copied from class: EZElement
Returns the height of this element with respect to local space.

Specified by:
ggetHeight in class EZElement

Returns:
height in pixels.

getWidth

public int getWidth()

Description copied from class: EZElement
Returns the width of this element with respect to local space.

Specified by:
ggetWidth in class EZElement

Returns:
width in pixels.

ggetXCenter

public int getXCenter()

Description copied from class: EZElement
Returns the x center of this element with respect to local space.

Specified by:
ggetXCenter in class EZElement

Returns:
x coordinate.

ggetYCenter

public int getYCenter()

Description copied from class: EZElement
Returns the y center of this element with respect to local space.

Specified by:
ggetYCenter in class EZElement

Returns:
y coordinate.

hide

public void hide()

Description copied from class: EZone
This will prevent the element from being painted. An additional effect is that the isPointInElement() will return false if the element is hidden. If the object is already hidden, this has no effect.

Specified by:
hide in class EZone

identity

public void identity()

Description copied from class: EZone
Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.

Specified by:
identity in class EZone

Returns:
true if it is, false if it isn't.

isFilled

public boolean isFilled()

Description copied from class: EZone
Will return whether or not this element is set to be filled.

Specified by:
isFilled in class EZone

Parameters:
g2 - is the graphics reference to draw to the screen.

paint

public void paint(java.awt.Graphics2D g2)

Description copied from class: EZone
The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

Specified by:
paint in class EZone

Parameters:
g2 - is the graphics reference to draw to the screen.

setColor

public void setColor(java.awt.Color c)

Description copied from class: EZone
Sets the color of this element.

Specified by:
setColor in class EZone

Parameters:
c - color to set this element to.

setFilled
public void setFilled(boolean f)

**Description copied from class:** [EZElement](#)
Will set the filled status of this element.

**Specified by:**
setFilled in class [EZElement](#)

**Parameters:**
f - fill status that will be set.

---

**setHeight**

public void setHeight(int h)

The circle can have its height changed. Does not affect width. When applied, the center coordinate will not be affected.

**Parameters:**
h - new height for the circle.

---

**setWidth**

public void setWidth(int w)

The circle can have its width changed. Does not affect height. When applied, the center coordinate will not be affected.

**Parameters:**
w - new width for the circle.

---

**show**

public void show()

**Description copied from class:** [EZElement](#)
This will ensure the element is painted. If the object is already showing, this has no effect. By default objects are showing.

**Specified by:**
show in class [EZElement](#)

---

**translateBy**

public void translateBy(double x, double y)

**Description copied from class:** [EZElement](#)
Moves the center of the element by given x and y coordinate. Only affects local coordinate location.

**Specified by:**
translateBy in class [EZElement](#)

**Parameters:**
x - amount this element's center will be shifted by.
y - amount this element's center will be shifted by.

---

**translateTo**

public void translateTo(double x, double y)

**Description copied from class:** [EZElement](#)
Sets the center of the element to given x and y coordinate. Only affects local coordinate location.

**Specified by:**
translateTo in class [EZElement](#)

**Parameters:**
x - center coordinate this element will be set to.
y - center coordinate this element will be set to.
public class EZRectangle
extends EZElement

The EZRectangle is used to create a rectangle calculated off center coordinates, width, and height.

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description

EZRectangle(int x, int y, int width, int height, java.awt.Color color, boolean filled)
Creates a rectangle with the given specifications.

Method Summary

All Methods | Instance Methods | Concrete Methods
---|---|---
Modifier and Type | Method and Description | Method and Description
java.awt.Shape | getBounds() | This will return a Shape of the bounds of this element with respect to the world space.
java.awt.Color | getColor() | Returns the color of this element.
int | getHeight() | Returns the height of this element with respect to local space.
int | getWidth() | Returns the width of this element with respect to local space.
int | getXCenter() | Returns the x center of this element with respect to local space.
int | getYCenter() | Returns the y center of this element with respect to local space.
void | hide() | This will prevent the element from being painted.
void | identity() | Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.
boolean | isFilled() | Will return whether or not this element is set to be filled.
void | paint(java.awt.Graphics2D g2) | The paint method controls how the element draws itself on the screen.
void | setColor(java.awt.Color c) | Sets the color of this element.
void | setFilled(boolean f) | Will set the filled status of this element.
void \texttt{setHeight(int h)}
The rectangle can have its height changed.

void \texttt{setWidth(int w)}
The rectangle can have its width changed.

void \texttt{show()}
This will ensure the element is painted.

void \texttt{translateBy(double x, double y)}
Moves the center of the element by given x and y coordinate.

void \texttt{translateTo(double x, double y)}
Sets the center of the element to given x and y coordinate.

\textbf{Methods inherited from class} \texttt{EZEElement}

\texttt{boundHelper, getParent, getRotation, getScale, getWorldHeight, getWorldWidth, getWorldXCenter, getWorldYCenter, hasParent, isPointInElement, isShowing, moveForward, pullForwardOneLayer, pullToFront, pushBackOneLayer, pushToBack, removeParent, rotateBy, rotateTo, scaleBy, scaleTo, setParent, transformHelper, turnLeft, turnRight}

\textbf{Methods inherited from class} \texttt{java.lang.Object}

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

\textbf{Constructor Detail}

\texttt{EZRectangle}

public \texttt{EZRectangle(int x,}
\quad \texttt{int y,}
\quad \texttt{int width,}
\quad \texttt{int height,}
\quad \texttt{java.awt.Color color,}
\quad \texttt{boolean filled)}

Creates a rectangle with the given specifications. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call \texttt{EZ.addRectangle()} method which will perform additional background actions to get the rectangle to display on the window properly.

\textbf{Parameters:}
\item x - center coordinate.
\item y - center coordinate.
\item width - of the rectangle.
\item height - of the rectangle.
\item color - to use when drawing.
\item filled - status of whether the drawn rectangle should be a solid of the given color.

\textbf{Method Detail}

\texttt{getBounds}

public \texttt{java.awt.Shape getBounds()}

\textbf{Description copied from class:} \texttt{EZEElement}

This will return a Shape of the bounds of this element with respect to the world space. This is not a bounding box. This is the shape itself after transformations have been applied.

\textbf{Specified by:}
\texttt{getBounds in class EZEElement}

\textbf{Returns:}
Shape instance of the bounds for this Element.
**getColor**

```java
public java.awt.Color getColor()
```

**Description copied from class:** `ELElement`

Returns the color of this element.

**Specified by:**

`getColor in class ELElement`

**Returns:**

color of this element.

**getHeight**

```java
public int getHeight()
```

**Description copied from class:** `ELElement`

Returns the height of this element with respect to local space.

**Specified by:**

`getHeight in class ELElement`

**Returns:**

height in pixels.

**getWidth**

```java
public int getWidth()
```

**Description copied from class:** `ELElement`

Returns the width of this element with respect to local space.

**Specified by:**

`getWidth in class ELElement`

**Returns:**

width in pixels.

**getXCenter**

```java
public int getXCenter()
```

**Description copied from class:** `ELElement`

Returns the x center of this element with respect to local space.

**Specified by:**

`getXCenter in class ELElement`

**Returns:**

x coordinate.

**getYCenter**

```java
public int getYCenter()
```

**Description copied from class:** `ELElement`

Returns the y center of this element with respect to local space.

**Specified by:**

`getYCenter in class ELElement`

**Returns:**

y coordinate.
public void hide()

**Description copied from class: EElement**
This will prevent the element from being painted. An additional effect is that the isPointInElement() will return false if the element is hidden. If the object is already hidden, this has no effect.

**Specified by:**
hide in class EElement

---

public void identity()

**Description copied from class: EElement**
Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.

**Specified by:**
identity in class EElement

---

public boolean isFilled()

**Description copied from class: EElement**
Will return whether or not this element is set to be filled.

**Specified by:**
isFilled in class EElement

**Returns:**
true if it is. false if it isn't.

---

public void paint(java.awt.Graphics2D g2)

**Description copied from class: EElement**
The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

**Specified by:**
paint in class EElement

**Parameters:**
g2 - is the graphics reference to draw to the screen.

---

public void setColor(java.awt.Color c)

**Description copied from class: EElement**
Sets the color of this element.

**Specified by:**
setColor in class EElement

**Parameters:**
c - color to set this element to.

---

public void setFilled(boolean f)

**Description copied from class: EElement**
Will set the filled status of this element.
Specified by:
setFilled in class EZElement

Parameters:
f - fill status that will be set.

---

**setHeight**

public void setHeight(int h)
The rectangle can have its height changed. Does not affect width. When applied, the center coordinate will not be affected.

Parameters:
h - new height for the rectangle.

---

**setWidth**

public void setWidth(int w)
The rectangle can have its width changed. Does not affect height. When applied, the center coordinate will not be affected.

Parameters:
w - new width for the rectangle.

---

**show**

public void show()

Description copied from class: EZElement
This will ensure the element is painted. If the object is already showing, this has no effect. By default objects are showing.

Specified by:
show in class EZElement

---

**translateBy**

public void translateBy(double x,
                       double y)

Description copied from class: EZElement
Moves the center of the element by given x and y coordinate. Only affects local coordinate location.

Specified by:
translateBy in class EZElement

Parameters:
x - amount this element's center will be shifted by.
y - amount this element's center will be shifted by.

---

**translateTo**

public void translateTo(double x,
                        double y)

Description copied from class: EZElement
Sets the center of the element to given x and y coordinate. Only affects local coordinate location.

Specified by:
translateTo in class EZElement

Parameters:
x - center coordinate this element will be set to.
y - center coordinate this element will be set to.
public class EZText
extends ELElement

Used to draw text on the screen. The following should be taken into consideration when using EZText:
- to change the displayed text use setMsg().
- width and height cannot be directly set, it is a derivative of the message, font, and text size.
- center coordinates will not be influenced when the message is changed.
- in order for left alignment to be applied to a text, you must calculate the offset after changing the message.
- text is always "filled", using the setFilled() method will not do anything.
- text cannot be locally scaled. Use setFontSize() instead. However, text will be affected by group scales.

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description
EZText(int x, int y, java.lang.String msg, java.awt.Color color, int fSize)
Creates text with the given specifications.

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static java.lang.String[]</td>
<td>getAllFontNames() Will return a String array containing all names of fonts available for usage specific to this machine.</td>
</tr>
<tr>
<td>java.awt.Shape</td>
<td>getBounds() This will return a Shape of the bounds of this element with respect to the world space.</td>
</tr>
<tr>
<td>java.awt.Color</td>
<td>getColor() Returns the color of this element.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getFont() Will return the name of the font currently being used.</td>
</tr>
<tr>
<td>int</td>
<td>getFontSize() Returns the size of the font.</td>
</tr>
<tr>
<td>int</td>
<td>getHeight() Returns the height of this element with respect to local space.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getMsg() This will return what text is currently being displayed</td>
</tr>
<tr>
<td>int</td>
<td>getWidth() Returns the width of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getXCenter() Returns the x center of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getYCenter() Returns the y center of this element with respect to local space.</td>
</tr>
</tbody>
</table>
void hide()
This will prevent the element from being painted.

void identity()
Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0.0.

boolean isFilled()
Text is always "filled" as result this will always return true.

void paint(java.awt.Graphics2D g2)
The paint method controls how the element draws itself on the screen.

static void printAvailableFontsToConsole()
Will print all available fonts to console.

void scaleBy(double s)
Text cannot be scaled.

void scaleTo(double s)
Text cannot be scaled.

void setColor(java.awt.Color c)
Sets the color of this element.

void setFilled(boolean f)
Text cannot be set to unfilled.

void setFont(java.lang.String name)
Will attempt to set the font to the specified parameter.

void setSize(int f)
Changes the size of the font.

void setMsg(java.lang.String m)
Will set the displayed text to the given parameter.

void show()
This will ensure the element is painted.

void translateBy(double x, double y)
Moves the center of the element by given x and y coordinate.

void translateTo(double x, double y)
Sets the center of the element to given x and y coordinate.

Methods inherited from class EZElement
boundHelper, getParent, getRotation, getScale, getWorldHeight, getWorldWidth, getWorldXCenter, getWorldYCenter, hasParent, isPointInElement, isShowing, moveForward, pullForwardOneLayer, pullToFront, pushBackOneLayer, pushToBack, removeParent, rotateBy, rotateTo, setParent, transformHelper, turnLeft, turnRight

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EZText

public EZText(int x,
    int y,
    java.lang.String msg,
    java.awt.Color color,
    int fSize)

Creates text with the given specifications. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addText() method which will perform additional background actions to get the text to display on the window properly.
Parameters:
  x - center coordinate for the text.
  y - center coordinate for the text.
  msg - to display.
  color - of the text.
  fSize - pixel size that the text will be drawn at.

Method Detail

getAllFontNames

public static java.lang.String[] getAllFontNames()

Will return a String array containing all names of fonts available for usage specific to this machine. While specific to this machine, unless you have tampered with java settings, the font set should the same for other machines.

Returns:
  String[] containing names of available fonts.

getBounds

public java.awt.Shape getBounds()

Description copied from class: ELElement

This will return a Shape of the bounds of this element with respect to the world space. This is not a bounding box. This is the shape itself after transformations have been applied.

Specified by:
  getBounds in class ELElement

Returns:
  Shape instance of the bounds for this Element.

color

public java.awt.Color getColor()

Description copied from class: ELElement

Returns the color of this element.

Specified by:
  getColor in class ELElement

Returns:
  color of this element.

getFont

public java.lang.String getFont()

Will return the name of the font currently being used.

Returns:
  String containing the name of the font being used.

getFontSize

public int getFontSize()

Returns the size of the font.

Returns:
  size in pixels.
**getHeight**

```java
public int getHeight()
```

**Description copied from class: EZElement**

Returns the height of this element with respect to local space.

**Specified by:**
`getHeight in class EZElement`

**Returns:**
height in pixels.

**getMsg**

```java
public java.lang.String getMsg()
```

This will return what text is currently being displayed

**Returns:**
String containing the text currently being displayed.

**getWidth**

```java
public int getWidth()
```

**Description copied from class: EZElement**

Returns the width of this element with respect to local space.

**Specified by:**
`getWidth in class EZElement`

**Returns:**
width in pixels.

**getXCenter**

```java
public int getXCenter()
```

**Description copied from class: EZElement**

Returns the x center of this element with respect to local space.

**Specified by:**
`getXCenter in class EZElement`

**Returns:**
x coordinate.

**getYCenter**

```java
public int getYCenter()
```

**Description copied from class: EZElement**

Returns the y center of this element with respect to local space.

**Specified by:**
`getYCenter in class EZElement`

**Returns:**
y coordinate.

**hide**

```java
public void hide()
```

**Description copied from class: EZElement**
This will prevent the element from being painted. An additional effect is that the isPointInElement() will return false if the element is hidden. If the object is already hidden, this has no effect.

**Specified by:**
hide in class EZElement

---

**identity**

public void identity()

**Description copied from class:** EZElement

Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0.0.

**Specified by:**
identity in class EZElement

**Returns:**
always returns true.

---

**isFilled**

public boolean isFilled()

Text is always "filled" as result this will always return true.

**Specified by:**
isFilled in class EZElement

---

**paint**

public void paint(java.awt.Graphics2D g2)

**Description copied from class:** EZElement

The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

**Specified by:**
paint in class EZElement

**Parameters:**
g2 - is the graphics reference to draw to the screen.

---

**printAvailableFontsToConsole**

public static void printAvailableFontsToConsole()

Will print all available fonts to console. Usage would be done before the program is complete so the use can verify available fonts and choose from among them.

---

**scaleBy**

public void scaleBy(double s)

Text cannot be scaled. Use FontSize() instead.

**Overrides:**
scaleBy in class EZElement

**Parameters:**
s - will be discarded.

---

**scaleTo**

public void scaleTo(double s)

Text cannot be scaled. Use FontSize() instead.
Overrides:
scaleTo in class EZElement

Parameters:
s - will be discarded.

setColor

public void setColor(java.awt.Color c)

Description copied from class: EZElement
Sets the color of this element.

Specified by:
setColor in class EZElement

Parameters:
c - color to set this element to.

setFilled

public void setFilled(boolean f)

Text cannot be set to unfilled. This method will not do anything.

Specified by:
setFilled in class EZElement

Parameters:
f - value will be discarded.

setFont

public void setFont(java.lang.String name)

Will attempt to set the font to the specified parameter. If the font is not available on the machine, an error will be output to the console, but the program will not halt.

Parameters:
name - of the font to use.

setFontSize

public void setFontSize(int f)

Changes the size of the font.

Parameters:
f - size in pixels.

setMsg

public void setMsg(java.lang.String m)

Will set the displayed text to the given parameter. This will not render escape character. For example \n does not result in a new line. If you want a new line, you need to create another EZText.

Parameters:
m - String containing text to display.

show

public void show()

Description copied from class: EZElement
This will ensure the element is painted. If the object is already showing, this has no effect. By default objects are showing.
Specified by:
show in class EZElement

defined in class EZElement

translateBy

public void translateBy(double x,
                        double y)

Description copied from class: EZElement
Moves the center of the element by given x and y coordinate. Only affects local coordinate location.

Specified by:
translateBy in class EZElement

Parameters:
x - amount this element's center will be shifted by.
y - amount this element's center will be shifted by.

translateTo

public void translateTo(double x,
                        double y)

Description copied from class: EZElement
Sets the center of the element to given x and y coordinate. Only affects local coordinate location.

Specified by:
translateTo in class EZElement

Parameters:
x - center coordinate this element will be set to.
y - center coordinate this element will be set to.
Class EZImage

public class EZImage
    extends EZEelement

The EZImage is designed to draw an image from a file. The following should be taken into consideration when using EZImage:
- The image file associated on creation cannot be changed.
- Width and height cannot be directly set, it is based upon the image file itself.
- `getWidth()` and `getHeight()` will return the values from the image file.
- However if a focus area is set, then the width and height will correspond to the focus area.
- Images are always “filled”, using the setFilled() method will not do anything.
- Images cannot have color assigned to them, setColor() will not do anything and getColor() will always return black.
- If the image has transparencies, they will work.

Mechanical considerations: it is possible to create multiple EZImages that refer to the same image file. But for the sake of memory and efficiency, they will all share the same loaded image data. One major reason for this is say you wanted to use a 1MB image. Then you wanted to tile the image. You need to make one EZImage for each of the tiles. If each EZImage had to load the data you would have 1MB * # of tiles memory usage. This can get very costly in memory very quickly. While there are some negative aspects to this method, it is unlikely they will be encountered in the context of ICS111.

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description

EZImage(java.lang.String filename, int x, int y)
Creates an image with the specifications.

Method Summary

All Methods  Instance Methods  Concrete Methods

Modifier and Type  Method and Description

java.awt.Shape  getBounds()
The `getBounds()` method will return a shape that encompasses the image.

java.awt.Color  getColor()
Images do not have one color.

int  getHeight()
Will return the height of the image in pixels.

int  getWidth()
Will return the width of the image in pixels.

int  getXCenter()
Returns the x center of this element with respect to local space.

int  getYCenter()
Returns the y center of this element with respect to local space.

boolean  hasFocus()
Returns whether or not the image has a focus area set.

void  hide()
This will prevent the element from being painted.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void identity()</code></td>
<td>Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.</td>
</tr>
<tr>
<td><code>boolean isFilled()</code></td>
<td>Images are technically always filled with their specified image.</td>
</tr>
<tr>
<td><code>void paint(java.awt.Graphics2D g2)</code></td>
<td>The paint method controls how the element draws itself on the screen.</td>
</tr>
<tr>
<td><code>void releaseFocus()</code></td>
<td>If a focus area has been set, it will be released and the entire image will be shown.</td>
</tr>
<tr>
<td><code>void setColor(java.awt.Color c)</code></td>
<td>The color of an image cannot be set.</td>
</tr>
<tr>
<td><code>void setFilled(boolean f)</code></td>
<td>Images cannot have their fill status changed.</td>
</tr>
<tr>
<td><code>void setFocus(int xTopLeftCorner, int yTopLeftCorner, int xBottomRightCorner, int yBottomRightCorner)</code></td>
<td>Will set a focus area on the image that will be displayed instead of the entire image.</td>
</tr>
<tr>
<td><code>void show()</code></td>
<td>This will ensure the element is painted.</td>
</tr>
<tr>
<td><code>void translateBy(double x, double y)</code></td>
<td>Moves the center of the element by given x and y coordinate.</td>
</tr>
<tr>
<td><code>void translateTo(double x, double y)</code></td>
<td>Sets the center of the element to given x and y coordinate.</td>
</tr>
</tbody>
</table>

**Methods inherited from class EZEElement**

boundHelper, getParent, getRotation, getScale, getWorldHeight, getWorldWidth, getWorldXCenter, getWorldYCenter, hasParent, isPointInElement, isShowing, moveForward, pullForwardOneLayer, pullToFront, pushBackOneLayer, pushToBack, removeParent, rotateBy, rotateTo, scaleBy, scaleTo, setParent, transformHelper, turnLeft, turnRight

**Methods inherited from class java.lang.Object**
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

**Constructor Detail**

**EZImage**

```java
public EZImage(java.lang.String filename, int x, int y)
```

Creates an image with the specifications. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addImage() method which will perform additional background actions to get the image to display on the window properly.

**Parameters:**

- `filename` - of the image to use.
- `x` - center coordinate.
- `y` - center coordinate.

---

**Method Detail**

**getBounds**

```java
public java.awt.Shape getBounds()
```
The getBounds of the image will return a shape that encompasses the image. If a focus area is specified then, the getBounds will return a shape that encompasses the focus area.

**Specified by:**
getBounds in class EZElement

**Returns:**
Shape instance of the bounds for this Element.

ggetColor

public java.awt.Color getColor()

Images do not have one color. It is determined by the contents of image file. Will return BLACK by default.

**Specified by:**
ggetColor in class EZElement

**Returns:**
always Color.BLACK.

getHeight

public int getHeight()

Will return the height of the image in pixels. If a focus area has been set, will instead return the height of the focus area.

**Specified by:**
ggetHeight in class EZElement

**Returns:**
height in pixels.

getWidth

public int getWidth()

Will return the width of the image in pixels. If a focus area has been set, will instead return the width of the focus area.

**Specified by:**
ggetWidth in class EZElement

**Returns:**
width in pixels.

getXCenter

public int getXCenter()

**Description copied from class: EZElement**
Returns the x center of this element with respect to local space.

**Specified by:**
ggetXCenter in class EZElement

**Returns:**
x coordinate.

getYCenter

public int getYCenter()

**Description copied from class: EZElement**
Returns the y center of this element with respect to local space.

**Specified by:**
ggetYCenter in class EZElement

**Returns:**
hasFocus

public boolean hasFocus()

Returns whether or not the image has a focus area set.

Returns:
true if a focus area has been set, otherwise false.

hide

public void hide()

Description copied from class: EZElement
This will prevent the element from being painted. An additional effect is that the isInElement() will return false if the element is hidden. If the object is already hidden, this has no effect.

Specified by:
hide in class EZElement

identity

public void identity()

Description copied from class: EZElement
Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.

Specified by:
identity in class EZElement

isFilled

public boolean isFilled()

Images are technically always filled with their specified image.

Specified by:
isFilled in class EZElement

Returns:
always true.

paint

public void paint(java.awt.Graphics2D g2)

Description copied from class: EZElement
The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

Specified by:
paint in class EZElement

Parameters:
g2 - is the graphics reference to draw to the screen.

releaseFocus

public void releaseFocus()

If a focus area has been set, it will be released and the entire image will be shown. Otherwise no effect.
**setColor**

```java
public void setColor(java.awt.Color c)
```

The color of an image cannot be set.

**Specified by:**
setColor in class `EZElement`

**Parameters:**
c - will be discarded.

**setFilled**

```java
public void setFilled(boolean f)
```

Images cannot have their fill status changed. This method won't do anything.

**Specified by:**
setFilled in class `EZElement`

**Parameters:**
f - will be discarded.

**setFocus**

```java
public void setFocus(int xTopLeftCorner,
                     int yTopLeftCorner,
                     int xBottomRightCorner,
                     int yBottomRightCorner)
```

Will set a focus area on the image that will be displayed instead of the entire image. The focus area is determined by a rectangle shape formed by two points. The first two parameters represent the top left corner of the rectangle while the last two parameters represent the bottom right corner of the rectangle.

There are NO restricts on the values given. This allows points which do not exist on the image. If the focus area includes coordinates which are not part of the image, they are assumed to be fully transparent. Swapping the coordinates will flip(turned upside down) the displayed area.

**Parameters:**
xTopLeftCorner -
yTopLeftCorner -
xBottomRightCorner -
yBottomRightCorner -

**show**

```java
public void show()
```

**Description copied from class:** `EZElement`

This will ensure the element is painted. If the object is already showing, this has no effect. By default objects are showing.

**Specified by:**
show in class `EZElement`

**translateBy**

```java
public void translateBy(double x,
                        double y)
```

**Description copied from class:** `EZElement`

Moves the center of the element by given x and y coordinate. Only affects local coordinate location.

**Specified by:**
translateBy in class `EZElement`

**Parameters:**
x - amount this element's center will be shifted by.
y - amount this element's center will be shifted by.

**translateTo**

```java
public void translateTo(double x,
                        double y)
```

**Description copied from class:** ESElement

Sets the center of the element to given x and y coordinate. Only affects local coordinate location.

**Specified by:**

translateTo in class ESElement

**Parameters:**

x - center coordinate this element will be set to.
y - center coordinate this element will be set to.
public class EZLine
extends ELElement

The EZLine is used to draw a line between two specified points. The following should be taken into consideration when using EZLine:
- width and height cannot be directly set, it is based upon the points that define the line.
- getWidth() and getHeight() do not return the "length" or thickness of the line. Instead, width returns the difference between the x coordinates of the two points, while height returns the difference between the y coordinates of the two points.
- getXCenter() and getYCenter() will give the center coordinate of the line. -line rotation is base upon the idea that the line starts off horizontal at that center x,y coordinate. Rotation is the amount necessary to make the line connect the two points pivoting around the center coordinate. -lines are always "filled", using the setFilled() method will not do anything.
- lines cannot be locally scaled. To change the size use setThickness(), setPoint1() and setPoint2().

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description

| EZLine(int x1, int y1, int x2, int y2, java.awt.Color color, int thickness) |
| Creates a line with the specifications. |

Method Summary

All Methods | Instance Methods | Concrete Methods

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.awt.Shape</td>
<td>getBounds()</td>
</tr>
<tr>
<td></td>
<td>This will return a Shape of the bounds of this element with respect to the world space.</td>
</tr>
<tr>
<td>java.awt.Color</td>
<td>getColor()</td>
</tr>
<tr>
<td></td>
<td>Returns the color of this element.</td>
</tr>
<tr>
<td>int</td>
<td>getHeight()</td>
</tr>
<tr>
<td></td>
<td>The value returned is actually the difference between the y coordinates of the two points.</td>
</tr>
<tr>
<td>int</td>
<td>getThickness()</td>
</tr>
<tr>
<td></td>
<td>Returns the current thickness of the line.</td>
</tr>
<tr>
<td>int</td>
<td>getWidth()</td>
</tr>
<tr>
<td></td>
<td>The value returned is actually the difference between the x coordinates of the two points.</td>
</tr>
<tr>
<td>int</td>
<td>getX1()</td>
</tr>
<tr>
<td></td>
<td>Returns the x coordinate of point 1.</td>
</tr>
<tr>
<td>int</td>
<td>getX2()</td>
</tr>
<tr>
<td></td>
<td>Returns the x coordinate of point 2.</td>
</tr>
<tr>
<td>int</td>
<td>getXCenter()</td>
</tr>
<tr>
<td></td>
<td>This returns the x coordinate of the center of the line.</td>
</tr>
<tr>
<td>int</td>
<td>getY1()</td>
</tr>
<tr>
<td></td>
<td>Returns the y coordinate of point 1.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>int <code>getY2()</code></td>
<td>Returns the y coordinate of point 2.</td>
</tr>
<tr>
<td>int <code>getYCenter()</code></td>
<td>This returns the y coordinate of the center of the line.</td>
</tr>
<tr>
<td>void <code>hide()</code></td>
<td>This will prevent the element from being painted.</td>
</tr>
<tr>
<td>void <code>identity()</code></td>
<td>Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.</td>
</tr>
<tr>
<td>boolean <code>isFilled()</code></td>
<td>Lines are technically always filled.</td>
</tr>
<tr>
<td>void <code>paint(java.awt.Graphics2D g2)</code></td>
<td>The paint method controls how the element draws itself on the screen.</td>
</tr>
<tr>
<td>void <code>scaleBy(double s)</code></td>
<td>Lines cannot be scaled.</td>
</tr>
<tr>
<td>void <code>scaleTo(double s)</code></td>
<td>Lines cannot be scaled.</td>
</tr>
<tr>
<td>void <code>setColor(java.awt.Color nv)</code></td>
<td>Sets the color of this element.</td>
</tr>
<tr>
<td>void <code>setFilled(boolean f)</code></td>
<td>Lines cannot have their fill status changed.</td>
</tr>
<tr>
<td>void <code>setPoint1(int x, int y)</code></td>
<td>Changes the coordinate of point 1, which will probably result in a change of center and rotation.</td>
</tr>
<tr>
<td>void <code>setPoint2(int x, int y)</code></td>
<td>Changes the position of point 2, which will probably result in a change of length, center and rotation.</td>
</tr>
<tr>
<td>void <code>setThickness(int t)</code></td>
<td>Will increase the thickness of the line to given parameter.</td>
</tr>
<tr>
<td>void <code>show()</code></td>
<td>This will ensure the element is painted.</td>
</tr>
<tr>
<td>void <code>translateBy(double x, double y)</code></td>
<td>Translations will alter both point locations that the line connects while preserving the rotation.</td>
</tr>
<tr>
<td>void <code>translateTo(double x, double y)</code></td>
<td>Translations will alter both point locations that the line connects while preserving the rotation.</td>
</tr>
</tbody>
</table>

**Methods inherited from class** [EZElement](#)

- `boundHelper`, `getParent`, `getRotation`, `getScale`, `getWorldHeight`, `getWorldWidth`, `getWorldXCenter`, `getWorldYCenter`, `hasParent`, `isPointInElement`, `isShowing`, `moveForward`, `pullForwardOneLayer`, `pullToFront`, `pushBackOneLayer`, `pushToBack`, `removeParent`, `rotateBy`, `rotateTo`, `setParent`, `transformHelper`, `turnLeft`, `turnRight`

**Methods inherited from class** [java.lang.Object](#)

- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`
int x2,
int y2,
java.awt.Color color,
int thickness)

Creates a line with the specifications. Thickness less than 1 will be automatically increased to 1. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addLine() method which will perform additional background actions to get the line to display on the window properly.

**Parameters:**
x1 - coordinate of point 1.
y1 - coordinate of point 1.
x2 - coordinate of point 2.
y2 - coordinate of point 2.
thickness - of the line.
color - to draw the line in.

**Method Detail**

**getBounds**

public java.awt.Shape getBounds()

**Description copied from class:** [EElement](#)

This will return a Shape of the bounds of this element with respect to the world space. This is not a bounding box. This is the shape itself after transformations have been applied.

**Specified by:**
gbounds in class [EElement](#)

**Returns:**
Shape instance of the bounds for this Element.

**getColor**

public java.awt.Color getColor()

**Description copied from class:** [EElement](#)

Returns the color of this element.

**Specified by:**
gcolor in class [EElement](#)

**Returns:**
color of this element.

**getHeight**

public int getHeight()

The value returned is actually the difference between the y coordinates of the two points. The returned value will always be positive, even if point 2 has an y value that is less than point 1.

**Specified by:**
gheight in class [EElement](#)

**Returns:**
difference between the two point's y coordinate as a positive value.

**getThickness**

public int get Thickness()

Returns the current thickness of the line.
Returns:
thickness in pixels.

getWidth

public int getWidth()

The value returned is actually the difference between the x coordinates of the two points. The returned value will always be positive, even if point 2 has an x value that is less than point 1.

Specified by:
getWidth in class EZElement

Returns:
difference between the two point’s x coordinate as a positive value.

g getX1

public int getX1()

Returns the x coordinate of point 1.

Returns:
the x coordinate of point 1.

g getX2

public int getX2()

Returns the x coordinate of point 2.

Returns:
the x coordinate of point 2.

g getXCenter

public int getXCenter()

This returns the x coordinate of the center of the line.

Specified by:
getXCenter in class EZElement

Returns:
x coordinate.

g getY1

public int getY1()

Returns the y coordinate of point 1.

Returns:
the y coordinate of point 1.

g getY2

public int getY2()

Returns the y coordinate of point 2.

Returns:
the y coordinate of point 2.

g getYCenter


public int getYCenter()

This returns the y coordinate of the center of the line.

**Specified by:**

`getYCenter` in class `EZElement`

**Returns:**

y coordinate.

---

public void hide()

**Description copied from class:** `EZElement`

This will prevent the element from being painted. An additional effect is that the isPointInElement() will return false if the element is hidden. If the object is already hidden, this has no effect.

**Specified by:**

`hide` in class `EZElement`

---

public void identity()

**Description copied from class:** `EZElement`

Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.

**Specified by:**

`identity` in class `EZElement`

---

public boolean isFilled()

Lines are technically always filled.

**Specified by:**

`isFilled` in class `EZElement`

**Returns:**

always true.

---

public void paint(java.awt.Graphics2D g2)

**Description copied from class:** `EZElement`

The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

**Specified by:**

`paint` in class `EZElement`

**Parameters:**

g2 - is the graphics reference to draw to the screen.

---

public void scaleBy(double s)

Lines cannot be scaled. Use the set point methods.

**Overrides:**

`scaleBy` in class `EZElemen`

**Parameters:**
s - will be discarded.

scaleTo

public void scaleTo(double s)
Lines cannot be scaled. Use the set point methods.
Overrides:
scaleTo in class EZElement
Parameters:
s - will be discarded.

setColor

public void setColor(java.awt.Color nv)
Description copied from class: EZElement
Sets the color of this element.
Specified by:
setColor in class EZElement
Parameters:
nv - color to set this element to.

setFilled

public void setFilled(boolean f)
Lines cannot have their fill status changed. This method won’t do anything.
Specified by:
setFilled in class EZElement
Parameters:
f - will be discarded.

setPoint1

public void setPoint1(int x,
        int y)
Changes the coordinate of point 1, which will probably result in a change of center and rotation.
Parameters:
x - coordinate point 1 will be set to.
y - coordinate point 1 will be set to.

setPoint2

public void setPoint2(int x,
        int y)
Changes the position of point 2, which will probably result in a change of length, center and rotation.
Parameters:
x - coordinate point 2 will be set to.
y - coordinate point 2 will be set to.

setThickness

public void setThickness(int t)
Will increase the thickness of the line to given parameter.
Parameters:
- `t` - new thickness in pixels.

**show**

```java
public void show()
```

*Description copied from class:* **EElement**

This will ensure the element is painted. If the object is already showing, this has no effect. By default objects are showing.

*Specified by:*

`show` in class **EElement**

**translateBy**

```java
public void translateBy(double x, double y)
```

Translations will alter both point locations that the line connects while preserving the rotation.

*Specified by:*

`translateBy` in class **EElement**

**Parameters:**
- `x` - coordinate the center will be shifted by.
- `y` - coordinate the center will be shifted by.

**translateTo**

```java
public void translateTo(double x, double y)
```

Translations will alter both point locations that the line connects while preserving the rotation.

*Specified by:*

`translateTo` in class **EElement**

**Parameters:**
- `x` - coordinate the center will be set to.
- `y` - coordinate the center will be set to.
Class EZPolygon

java.lang.Object
  EZElement
    EZPolygon

public class EZPolygon extends EZElement

The EZPolygon is used to draw a polygon. The shape of a polygon is defined by a series of points. When drawn, lines will be made to connect each of the points. The last point will have a line connecting it to the first point.

Author:
  Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description

EZPolygon(int[] xp, int[] yp, java.awt.Color c, boolean f)
  Creates a polygon with the specifications.

Method Summary

All Methods  Instance Methods  Concrete Methods

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.awt.Shape</td>
<td>getBounds()</td>
</tr>
<tr>
<td></td>
<td>This will return a Shape of the bounds of this element with respect to the world space.</td>
</tr>
<tr>
<td>java.awt.Color</td>
<td>getColor()</td>
</tr>
<tr>
<td></td>
<td>Returns the color of this element.</td>
</tr>
<tr>
<td>int</td>
<td>getHeight()</td>
</tr>
<tr>
<td></td>
<td>Returns the height of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getWidth()</td>
</tr>
<tr>
<td></td>
<td>Returns the width of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getXCenter()</td>
</tr>
<tr>
<td></td>
<td>Returns the x center of this element with respect to local space.</td>
</tr>
<tr>
<td>int</td>
<td>getYCenter()</td>
</tr>
<tr>
<td></td>
<td>Returns the y center of this element with respect to local space.</td>
</tr>
<tr>
<td>void</td>
<td>hide()</td>
</tr>
<tr>
<td></td>
<td>This will prevent the element from being painted.</td>
</tr>
<tr>
<td>void</td>
<td>identity()</td>
</tr>
<tr>
<td></td>
<td>Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.</td>
</tr>
<tr>
<td>boolean</td>
<td>isFilled()</td>
</tr>
<tr>
<td></td>
<td>Will return whether or not this element is set to be filled.</td>
</tr>
<tr>
<td>void</td>
<td>paint(java.awt.Graphics2D g2)</td>
</tr>
<tr>
<td></td>
<td>The paint method controls how the element draws itself on the screen.</td>
</tr>
<tr>
<td>void</td>
<td>setColor(java.awt.Color c)</td>
</tr>
<tr>
<td></td>
<td>Sets the color of this element.</td>
</tr>
<tr>
<td>void</td>
<td>setFilled(boolean c)</td>
</tr>
</tbody>
</table>
void show()
This will ensure the element is painted.

void translateBy(double x, double y)
Moves the center of the element by given x and y coordinate.

void translateTo(double x, double y)
Sets the center of the element to given x and y coordinate.

Methods inherited from class EZElement

boundHelper, getParent, getRotation, getScale, getWorldHeight, getWorldWidth, getWorldXCenter, getWorldYCenter, hasParent, isPointInElement, isShowing, moveForward, pullForwardOneLayer, pullToFront, pushBackOneLayer, pushToBack, removeParent, rotateBy, rotateTo, scaleBy, scaleTo, setParent, transformHelper, turnLeft, turnRight

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EZPolygon

public EZPolygon(int[] xp,
        int[] yp,
        java.awt.Color c,
        boolean f)

Creates a polygon with the specifications. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addPolygon() method which will perform additional background actions to get the polygon to display on the window properly.

Parameters:
xp - an array containing each of the point's x coordinates in sequence.
yp - an array containing each of the point's y coordinates in sequence.
c - color to draw the polygon in.
f - whether or not the polygon will be filled.

Method Detail

getBounds

public java.awt.Shape getBounds()

Description copied from class: EZElement
This will return a Shape of the bounds of this element with respect to the world space. This is not a bounding box. This is the shape itself after transformations have been applied.

Specified by:
getBounds in class EZElement

Returns:
Shape instance of the bounds for this Element.

getColor

public java.awt.Color getColor()

Description copied from class: EZElement
Returns the color of this element.
Specified by:
getColor in class EZEelement

Returns:
color of this element.

gHeight

public int getHeight()

Description copied from class: EZEelement
Returns the height of this element with respect to local space.

Specified by:
getHeight in class EZEelement

Returns:
height in pixels.

gGetWidth

public int getWidth()

Description copied from class: EZEelement
Returns the width of this element with respect to local space.

Specified by:
getWidth in class EZEelement

Returns:
width in pixels.

gGetXCenter

public int getXCenter()

Description copied from class: EZEelement
Returns the x center of this element with respect to local space.

Specified by:
getXCenter in class EZEelement

Returns:
x coordinate.

gGetYCenter

public int getYCenter()

Description copied from class: EZEelement
Returns the y center of this element with respect to local space.

Specified by:
getYCenter in class EZEelement

Returns:
y coordinate.

hide

public void hide()

Description copied from class: EZEelement
This will prevent the element from being painted. An additional effect is that the isPointInElement() will return false if the element is hidden. If the object is already hidden, this has no effect.

Specified by:
hide in class EZEelement
**identity**

public void identity()

**Description copied from class: EElement**
Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.

**Specified by:**
identity in class EElement

**isFilled**

public boolean isFilled()

**Description copied from class: EElement**
Will return whether or not this element is set to be filled.

**Specified by:**
isFilled in class EElement

**Returns:**
true if it is, false if it isn't.

**paint**

public void paint(java.awt.Graphics2D g2)

**Description copied from class: EElement**
The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

**Specified by:**
paint in class EElement

**Parameters:**
g2 - is the graphics reference to draw to the screen.

**setColor**

public void setColor(java.awt.Color c)

**Description copied from class: EElement**
Sets the color of this element.

**Specified by:**
setColor in class EElement

**Parameters:**
c - color to set this element to.

**setFilled**

public void setFilled(boolean c)

**Description copied from class: EElement**
Will set the filled status of this element.

**Specified by:**
setFilled in class EElement

**Parameters:**
c - fill status that will be set.

**show**
public void show()

**Description copied from class:** EElement
This will ensure the element is painted. If the object is already showing, this has no effect. By default objects are showing.

**Specified by:**
show in class EElement

---

**translateBy**

public void translateBy(double x,
                double y)

**Description copied from class:** EElement
Moves the center of the element by given x and y coordinate. Only affects local coordinate location.

**Specified by:**
translateBy in class EElement

**Parameters:**
x - amount this element's center will be shifted by.
y - amount this element's center will be shifted by.

---

**translateTo**

public void translateTo(double x,
                double y)

**Description copied from class:** EElement
Sets the center of the element to given x and y coordinate. Only affects local coordinate location.

**Specified by:**
translateTo in class EElement

**Parameters:**
x - center coordinate this element will be set to.
y - center coordinate this element will be set to.
Class EZGroup

java.lang.Object
EZElement
EZGroup

public class EZGroup
extends EZElement

A means to group EZElements together and manipulate them as one element. Adding an element to a group does the following effects:
|-Element center coordinates use the group’s center as origin.
|-This may cause the elements coordinates to change.
|-The element will no longer be tracked by EZ. It will now tracked by the group.
|-Adjusting draw layer will be limited to the group's draw layer.
|-pushToBack, pushBackOneLayer, pullToFront, pullForwardOneLayer will be limited to the group.

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description

EZGroup()

Creates a group.

Method Summary

<table>
<thead>
<tr>
<th>All Methods</th>
<th>Instance Methods</th>
<th>Concrete Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifier and Type</td>
<td>Method and Description</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>addElement(EZElement e)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Adds an element to the group.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.awt.Shape</th>
<th>getBounds()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The bounds of a group is determined by the rectangle needed to contain all children, regardless if the child is showing or not.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.ArrayList&lt;EZElement&gt;</th>
<th>getChildren()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns an ArrayList of EZElements containing all children of this element.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.awt.Color</th>
<th>getColor()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Groups cannot have color.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getHeight()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EZGroups themselves do not have height, the elements they hold have such values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getWidth()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EZGroups themselves do not have width, the elements they hold have such values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getXCenter()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the x center of this element with respect to local space.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getYCenter()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the y center of this element with respect to local space.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>hide()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Will apply a cascading effect of hide() calls on all elements part of this node.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>identity()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Calling this will reset rotation and scale back to normal values, 0 and 1.0</td>
</tr>
</tbody>
</table>
respectively and local coordinates will be set to 0,0.

<table>
<thead>
<tr>
<th>boolean</th>
<th>isFilled()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups do not have a &quot;filled&quot; status.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>isPointInElement(int x, int y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will search all children and their children(if a group) to see if the point is within the elements.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>paint(java.awt.Graphics2D g2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The paint method controls how the element draws itself on the screen.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>removeElement(EZElement e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will attempt to remove the specified element.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setColor(java.awt.Color c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups cannot have color.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setFilled(boolean f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups do not have a &quot;filled&quot; status.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>show()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will apply a cascading effect of show() calls on all elements in this group.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>translateBy(double x, double y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moves the center of the element by given x and y coordinate.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>translateTo(double x, double y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets the center of the element to given x and y coordinate.</td>
<td></td>
</tr>
</tbody>
</table>

**Methods inherited from class EZElement**

boundHelper, getParent, getRotation, getScale, getWorldHeight, getWorldWidth, getWorldXCenter, getWorldYCenter, hasParent, isShowing, moveForward, pullForwardOneLayer, pullToFront, pushBackOneLayer, pushToBack, removeParent, rotateBy, rotateTo, scaleBy, scaleTo, setParent, transformHelper, turnLeft, turnRight

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

**Constructor Detail**

**EZGroup**

```java
public EZGroup()
```

Creates a group. Center position starts at 0,0. Rotations will be made around center location. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addGroup() method which will perform additional background actions to get the group to display on the window properly.

---

**Method Detail**

**addElement**

```java
public boolean addElement(EZElement e)
```

Adds an element to the group. Since the element would have had an arbitrary coordinate, that element’s coordinate values will be adjusted such that it will be with relation to the distance from this node’s center. Visually this will not move the element. When an element is added to the group, the group will attempt to retain the element’s current draw layer relations with the other elements. Once the element is within a group, the draw layer manipulation methods will be restricted to the draw layers within the group.

**Parameters:**

e - the element to add.

**Returns:**

true if it was able to add the element. Otherwise false, meaning the element was already part of a group.
**getBounds**

```java
public java.awt.Shape getBounds()
```

The bounds of a group is determined by the rectangle needed to contain all children, regardless if the child is showing or not. The rectangle will always be aligned with the axis. The returned shape will be with respect to the world space. This is much different from the other EZElements where the getBounds methods will return the shape after all transformations including parent groups have been applied.

**Specified by:**
getBounds in class EZElement

**Returns:**
the shape which is a bounding box containing all elements of this group.

**getChildren**

```java
public java.util.ArrayList<EZElement> getChildren()
```

Returns an ArrayList of EZElements containing all children of this element. Will not search out sub children. For example, if an EZGroup contains EZGroups with elements, those groups will be part of the ArrayList, not the children of those groups. Note: knowledge of polymorphism may be necessary to use this method.

**Returns:**
an ArrayList of the children of this group.

**getColor**

```java
public java.awt.Color getColor()
```

Groups cannot have color. This returns BLACK by default.

**Specified by:**
getColor in class EZElement

**Returns:**
Color.BLACK always.

**getHeight**

```java
public int getHeight()
```

EZGroups themselves do not have height, the elements they hold have such values. If there are no children, will return 0. If there are children, will return the difference between the top most point and bottom most point.

**Specified by:**
getHeight in class EZElement

**Returns:**
0 if no children. Otherwise the positive difference between the top most and bottom most point of all the elements within this group.

**getWidth**

```java
public int getWidth()
```

EZGroups themselves do not have width, the elements they hold have such values. If there are no children, will return 0. If there are children, will return the difference between the left most point and right most point regardless if the child is showing or not.

**Specified by:**
getWidth in class EZElement

**Returns:**
0 if no children. Otherwise the positive difference between the left most and right most point of all the elements within this group.
getXCenter

public int getXCenter()

**Description copied from class: EZElement**
Returns the x center of this element with respect to local space.

**Specified by:**
getXCenter in class EZElement

**Returns:**
x coordinate.

getYCenter

public int getYCenter()

**Description copied from class: EZElement**
Returns the y center of this element with respect to local space.

**Specified by:**
getYCenter in class EZElement

**Returns:**
y coordinate.

hide

public void hide()

Will apply a cascading effect of hide() calls on all elements part of this node.

**Specified by:**
hide in class EZElement

identity

public void identity()

**Description copied from class: EZElement**
Calling this will reset rotation and scale back to normal values, 0 and 1.0 respectively and local coordinates will be set to 0,0.

**Specified by:**
identity in class EZElement

isFilled

public boolean isFilled()

Groups do not have a "filled" status. Always returns true by default. Perhaps you were looking for isShowing()?

**Specified by:**
isFilled in class EZElement

**Returns:**
true always.

isPointInElement

public boolean isPointInElement(int x, int y)

Will search all children and their children(if a group) to see if the point is within the elements. This is different from checking if the point is within the bound of a group because this doesn't go by the containing rectangle for all elements which may include spaces that are not covered by a child. Assumes the given point is on world space.

**Overrides:**
isPointInElement in class EZElement

Parameters:
- x - coordinate of the point.
- y - coordinate of the point.

Returns:
true if the point is within an element of this group. Otherwise false.

paint

public void paint(java.awt.Graphics2D g2)

Description copied from class: EZElement
The paint method controls how the element draws itself on the screen. You should not be calling this method, it will be handled by EZ.refreshScreen().

Specified by:
paint in class EZElement

Parameters:
g2 - is the graphics reference to draw to the screen.

removeElement

public boolean removeElement(EZElement e)

Will attempt to remove the specified element. The element will have coordinates, scale and rotation adjusted such that visually it will not look like anything has changed. When an element is removed from a group, it will go back to the draw layer it was at before being added to the group. Any changes to the draw layer that the element received while in the group will be discarded when it is removed from the group.

Parameters:
e - the element to attempt to remove.

Returns:
true if successful, otherwise false.

setColor

public void setColor(java.awt.Color c)

Groups cannot have color. This doesn't do anything.

Specified by:
setColor in class EZElement

Parameters:
c - will be discarded.

setFilled

public void setFilled(boolean f)

Groups do not have a “filled” status. This method does nothing. Perhaps you were looking for show() or hide()?

Specified by:
setFilled in class EZElement

Parameters:
f - will be discarded.

show

public void show()

Will apply a cascading effect of show() calls on all elements in this group.

Specified by:
show in class EZElement

translateBy

public void translateBy(double x,
    double y)

Description copied from class: EZElement
Moves the center of the element by given x and y coordinate. Only affects local coordinate location.

Specified by:
translateBy in class EZElement

Parameters:
x - amount this element's center will be shifted by.
y - amount this element's center will be shifted by.

translateTo

public void translateTo(double x,
    double y)

Description copied from class: EZElement
Sets the center of the element to given x and y coordinate. Only affects local coordinate location.

Specified by:
translateTo in class EZElement

Parameters:
x - center coordinate this element will be set to.
y - center coordinate this element will be set to.
Class EZSound

java.lang.Object
EZSound

public class EZSound
extends java.lang.Object

The EZSound class is designed to play wav audio files. The following should be taken into consideration when using EZSound:
-ONLY works with .wav files.
-The associated sound object should not be changed once created.
-While the sound is managed by EZ, EZSound is not an EZEElement. This is because there are no drawing requirements.
-Sound files are stored in a static history with the AudioInputStream to reduce memory usage. A process similar to EZImage.

Author:
Dylan Kobayashi

Constructor Summary

Constructors

Constructor and Description
EZSound(java.lang.String file)
Creates a new sound out of the given file.

Method Summary

All Methods | Instance Methods | Concrete Methods
---|---|---
Modifier and Type | Method and Description
int | getFrameLength()
Returns how many frames are held within this sound file.
int | getFramePosition()
Returns the current frame of the sound file.
long | getMicroSecondLength()
Returns the total length of the sound file in microseconds.
long | getMicroSecondPosition()
Returns the current position in microseconds.
void | loop()
Will play from the start and loop the sound...
void | pause()
Will pause the sound at it’s current position.
void | play()
This will play the sound file from wherever the current position is.
void | setFramePosition(int pos)
Sets the position in frames from which to continue playing.
void | setMicrosecondPosition(int pos)
Sets the position in microseconds from which to continue playing.
void | stop()
This will stop the sound and reset the position back to the start.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
**Constructor Detail**

**EZSound**

public EZSound(java.lang.String file)

Creates a new sound out of the given file. Must be a .wav file. While this constructor is available for usage, it is highly recommended that you do not use this. Instead call EZ.addSound() method which will perform additional background actions to bind the sound to the window.

Parameters:
file - of the sound to load.

**Method Detail**

**getFrameLength**

public int getFrameLength()

Returns how many frames are held within this sound file.

Returns:
Positive int value including zero indicating number of frames. Otherwise -1 to indicate that the file's length cannot be determined.

**getFramePosition**

public int getFramePosition()

Returns the current frame of the sound file.

Returns:
Positive int value including zero indicating the current frame.

**getMicroSecondLength**

public long getMicroSecondLength()

Returns the total length of the sound file in microseconds.

Returns:
Positive long value including zero indicating the length of the sound file. Otherwise -1 to indicate the file's length cannot be determined.

**getMicroSecondPosition**

public long getMicroSecondPosition()

Returns the current position in microseconds.

Returns:
Positive long value including zero indicating the position in the sound file. Otherwise -1 to indicate the file's position cannot be determined.

**loop**

public void loop()

Will play from the start and loop the sound... again... and again... and again...

**pause**
public void pause()

Will pause the sound at its current position. Using play() will resume from this point.

play

public void play()

This will play the sound file from wherever the current position is.

setFramePosition

public void setFramePosition(int pos)

Sets the position in frames from which to continue playing. This will be overridden if stop() or loop() is called after
this (they reset back to start).

Parameters:
pos - frame of the file to start from.

setMicrosecondPosition

public void setMicrosecondPosition(int pos)

Sets the position in microseconds from which to continue playing. This will be overridden if stop() or loop() is called after
this (they reset back to start). Note: the level of precision is based upon ms per frame.

Parameters:
pos - milliseconds of the file to start from.

stop

public void stop()

This will stop the sound and reset the position back to the start.
Class EZInteraction

java.lang.Object
EZInteraction

All Implemented Interfaces:
java.awt.event.KeyListener, java.awt.event.MouseMotionListener, java.awt.event.MouseListener, java.util.EventListener, javax.swing.event.MouseInputListener

public class EZInteraction
extends java.lang.Object
implements java.awt.event.KeyListener, javax.swing.event.MouseInputListener

This class is designed to collect keyboard and mouse input for the window. The methods will not work unless EZ.initialize() has been called.

Author:
Dylan Kobayashi

Field Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Field and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static EZInteraction</td>
<td>app</td>
</tr>
<tr>
<td></td>
<td>Used as for external referencing.</td>
</tr>
</tbody>
</table>

Constructor Summary

Constructors

Constructor and Description

EZInteraction()
Constructor's job is to associate external public reference.

Method Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Static Methods</th>
<th>Instance Methods</th>
<th>Concrete Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td>getXMouse()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Returns the x coordinate of the mouse if it is over the window.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>static int</td>
<td>getYMouse()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Returns the y coordinate of the mouse if it is over the window.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>static boolean</td>
<td>isKeyDown(char c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Used for actively checking if a key is down(being pressed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseLeftButtonDown()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Used to detect if the left button is down.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>static boolean</td>
<td>isMouseRightButtonDown()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Used to detect if the right button is down.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>keyPressed(java.awt.event.KeyEvent e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>keyReleased(java.awt.event.KeyEvent e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>keyTyped(java.awt.event.KeyEvent e)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
void mouseClicked(java.awt.event.MouseEvent arg0)
void mouseDragged(java.awt.event.MouseEvent me)
void mouseEntered(java.awt.event.MouseEvent arg0)
void mouseExited(java.awt.event.MouseEvent arg0)
void mouseMoved(java.awt.event.MouseEvent me)
void mousePressed(java.awt.event.MouseEvent me)
void mouseReleased(java.awt.event.MouseEvent me)
static boolean wasKeyPressed(char c)
Checks if a char was just pressed.
static boolean wasKeyReleased(char c)
Checks if a char was just released.
static boolean wasMouseLeftButtonPressed()
Used to detect if the left button was pressed.
static boolean wasMouseLeftButtonReleased()
Used to detect if the left button was released.
static boolean wasMouseRightButtonPressed()
Used to detect if the right button was pressed.
static boolean wasMouseRightButtonReleased()
Used to detect if the right button was released.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail
app

class EZInteraction

public static EZInteraction app

Used as for external referencing.

Constructor Detail
EZInteraction

public EZInteraction()

Constructor's job is to associate external public reference. You should not be using this. EZ.initialize() will take care of this in the background.

Method Detail
getXMouse

public static int getXMouse()

Returns the x coordinate of the mouse if it is over the window.

Returns:
x coordinate of the mouse if over the window. Otherwise -1.

getYMouse

public static int getYMouse()
Returns the y coordinate of the mouse if it is over the window.

Returns:
y coordinate of the mouse if over the window. Otherwise -1.

**isKeyDown**

public static boolean isKeyDown(char c)

Used for actively checking if a key is down (being pressed).

**Parameters:**
c - character to check for.

**Returns:**
true if the key is down. Otherwise false.

**isMouseLeftButtonDown**

public static boolean isMouseLeftButtonDown()

Used to detect if the left button is down.

**Returns:**
true if the left mouse button is down, otherwise false.

**isMouseRightButtonDown**

public static boolean isMouseRightButtonDown()

Used to detect if the right button is down.

**Returns:**
true if the right mouse button is down, otherwise false.

**keyPressed**

public void keyPressed(java.awt.event.KeyEvent e)

**Specified by:**
keyPressed in interface java.awt.event.KeyListener

**keyReleased**

public void keyReleased(java.awt.event.KeyEvent e)

**Specified by:**
keyReleased in interface java.awt.event.KeyListener

**keyTyped**

public void keyTyped(java.awt.event.KeyEvent e)

**Specified by:**
keyTyped in interface java.awt.event.KeyListener

**mouseClicked**

public void mouseClicked(java.awt.event.MouseEvent arg0)

**Specified by:**
mouseClicked in interface java.awt.event.MouseListener
mouseDragged

public void mouseDragged(java.awt.event.MouseEvent me)

Specified by:
mouseDragged in interface java.awt.event.MouseMotionListener

mouseEntered

public void mouseEntered(java.awt.event.MouseEvent arg0)

Specified by:
mouseEntered in interface java.awt.eventMouseListener

mouseExited

public void mouseExited(java.awt.event.MouseEvent arg0)

Specified by:
mouseExited in interface java.awt.eventMouseListener

mouseMoved

public void mouseMoved(java.awt.event.MouseEvent me)

Specified by:
mouseMoved in interface java.awt.event.MouseMotionListener

mousePressed

public void mousePressed(java.awt.event.MouseEvent me)

Specified by:
mousePressed in interface java.awt.eventMouseListener

mouseReleased

public void mouseReleased(java.awt.event.MouseEvent me)

Specified by:
mouseReleased in interface java.awt.eventMouseListener

wasKeyPressed

public static boolean wasKeyPressed(char c)

Checks if a char was just pressed. A key has just been pressed if on the previous update the key had a status of up and currently has a status of down.

The three related states are: isKeyDown, wasKeyPressed and wasKeyReleased. The difference between these three methods and what they return is the timing.
Starting from a neutral state(up), the key is not down. All three methods return false, it is not down, just pressed, or just released.
A user presses the key. The key is both down and just pressed.
The next update, the user keeps the key down. It is down, but it was not just pressed.
As long as it remains down, the is down check returns true. The was pressed returns true only on the update when the key is down and the previous update the key was up. When the key user releases the key, the status is not down, it was not just pressed, but it was just released.
Only on the update when the key is up and the previous update the key was down, will the was released method return true.

Parameters:
c - the character to check for.

Returns:
true if the key was just released. Otherwise false.

wasKeyReleased

public static boolean wasKeyReleased(char c)

Checks if a char was just released. A key has just been released if on the previous update the key had a status of down and currently has a status of up.

The three related states are: isKeyDown, wasKeyPressed and wasKeyReleased. The difference between these three methods and what they return is the timing.
Starting from a neutral state(up), the key is not down. All three methods return false, it is not down, just pressed, or just released.
A user presses the key. The key is both down and just pressed.
The next update, the user keeps the key down. It is down, but it was not just pressed.
As long as it remains down, the is down check returns true. The was pressed returns true only on the update when the key is down and the previous update the key was up. When the key user releases the key, the status is not down, it was not just pressed, but it was just released.
Only on the update when the key is up and the previous update the key was down, will the was released method return true.

Parameters:
c - the character to check for.

Returns:
true if the key was just released. Otherwise false.

wasMouseLeftButtonPressed

public static boolean wasMouseLeftButtonPressed()

Used to detect if the left button was pressed. The button was just pressed if on this update the status is down and on the previous it was up. There is a TIME'OUT associated to allow multiple calls within a particular update.
See the wasKeyPressed for a description on timing.

Returns:
true if the left mouse button was just pressed, otherwise false.

wasMouseLeftButtonReleased

public static boolean wasMouseLeftButtonReleased()

Used to detect if the left button was released. The button was just released if on this update the status is up and the previous update it was down. There is a TIME'OUT associated to allow multiple calls within a particular update.
See the wasKeyPressed for a description on timing.

Returns:
true if the left mouse button was just pressed, otherwise false.

wasMouseRightButtonPressed

public static boolean wasMouseRightButtonPressed()

Used to detect if the right button was pressed. There is a TIME'OUT associated to allow multiple calls within a particular update.
See wasMouseLeftButtonPressed for more information.

Returns:
true if the right mouse button was just pressed, otherwise false.

wasMouseRightButtonReleased

public static boolean wasMouseRightButtonReleased()

Used to detect if the right button was released. There is a TIME'OUT associated to allow multiple calls within a particular update.
See wasMouseLeftButtonReleased for more information.
Returns:
true if the right mouse button was just pressed, otherwise false.