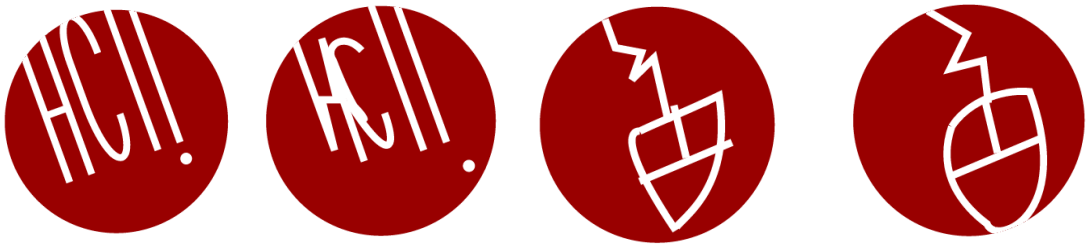
HCI-5/433D Web Lab
Project 2:
Your Own <canvas> Drawing Library
Due: Thursday, October 3, 2013 by 10:00 am



Goal
The goal of this project is to create a re-usable drawing library for creating interactive content on the `<canvas>` tag. Some pre-existing libraries are http://raphaeljs.com, http://processingjs.org, fabric.js (http://kangax.github.com/fabric.js/kitchensink/), and jsdraw2d (http://jsdraw2d.jsfiction.com/). For this project, you're going to write your own drawing library, and then use this library to create an interesting doodle of your own!

Files Provided
The following files are in the same .zip folder as this project description was in:

`doodle-library- skeleton.js`	Skeleton code for the JavaScript library you will be creating.
`utils.js`	Utility functions that have been provided for you.
`primitive-test.html`	HTML file that is provided for you to test your library.
`primitive-test.js`	JavaScript file that is provided for you to test your library.
`container-test.html`	HTML file that is provided for you to test your library.
`container-test.js`	JavaScript file that is provided for you to test your library.
`primitives.png`	Screenshot of test output for first part of tests (i.e. simple primitives)
`containers.png`	Screenshot of text output for second part of tests (i.e. complex containers)

Project Overview

This project has two parts. The first part is to create a drawing library for <canvas> that provides a few basic functions to make drawing using canvas easier. The second part is to use your library to create a cool doodle (ideas at http://www.google.com/logos/).

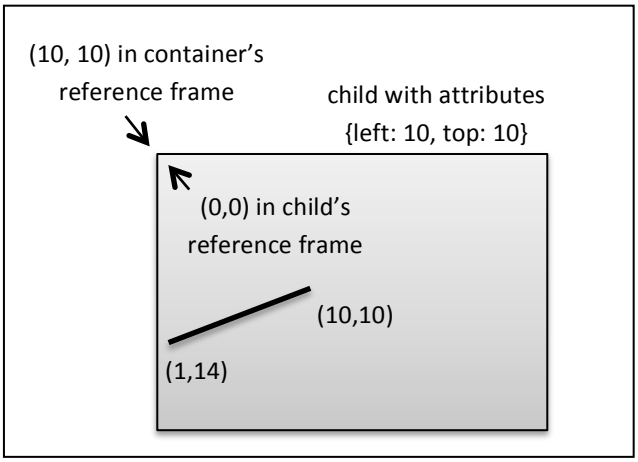
Part I: Drawing Library

Your drawing library must implement the objects specified below. Your objects should have at least the methods and attributes specified below (though you may add more if you wish). The utils.js file provides a mechanism for executing inheritance, as well as other useful functions for setting default values. Inheritance has already been set up for you in the skeleton code, and you can use it for creating new objects for extra credit.

Your objects will need to follow a specific layout protocol. **There is a root entity (a Doodle) which represents the <canvas> element that you're drawing on**. Its purpose is to hold and draw all of the elements in your doodle. It cannot be rotated or translated.

Every drawable object in a doodle has *left* and *top* coordinates. Left and top specify the reference frame for the object; **when the object draws itself, it treats its left and top as the origin, (0, 0)**. Therefore, x and y coordinates in the attributes for an object, such as a line or a path, are relative to the object's left and top.

For example, the figure below shows a container with a single child container. The child container has a left and top of (10, 10), so it is drawn at (10, 10) in the parent's coordinate system. The line inside the child container, however, treats the left, top point of the container as (0, 0) when drawing its endpoints.



When a container object (such as a Doodle or Container object) draws one of its child elements, it first translates and rotates the child element according to the child's left and top attributes. It then calls the child's draw function.

The following are specifications for the objects you must implement. Stubs for these objects are provided in the starter code.

Doodle

A Doodle is the root container for all drawable elements. It represents the canvas on which all other elements will be drawn. When drawing using this library, the first element created is an instance of this Doodle element. Additional elements are added as children to the root Doodle element.

Constructor Parameter:

***context*:** The drawing context for the object

Methods:

draw: Draws its children. If a child is not visible, does not draw the child.

Fields:

context: The drawing context for the object, generated by a canvas

children: An array of the top-level drawable elements in this doodle.

***Hint: The drawing context is stored in Doodle (the canvas object) and can be passed to methods of a child element to do drawing.**

Text

Inherits from: Drawable

A Text object draws text with the given attributes. Note: Text is always drawn up from the *bottom* of the object. This is because there is no simple way to measure the height of a text string using canvas, so we explicitly define it with height.

Constructor Parameter:

attrs: An object containing values for each of the fields in the object. If the ***attrs*** parameter is not specified, or if one of the fields is not specified, use predefined defaults (defined for you).

Methods:

draw: Draw the text using the values defined in attrs.

Fields (in addition to those specified by ancestors):

content: The text string to draw.

fill: The fill color of the text. Specify in the form of a CSS color.

font: The font style of the text. You can again use the CSS font specification.

height: The height of your text, in pixels.

DoodleImage

Inherits from: Drawable

A DoodleImage object draws an image.

Constructor Parameter:

attrs: An object containing values for each of the fields in the object. If the ***attrs*** parameter is not specified, or if one of the fields is not specified, use predefined defaults (defined for you).

Methods:

draw: Draw the image using the specified source, with the specified width and height.

Fields (in addition to those specified by ancestors):

width: The width of the image. Default is -1. If image width is -1, use the natural width of the image.

height: The height of the image. Default is -1. If image height is -1, use the natural height of the image.

src: The location of the image (specify a path).

Line

Inherits from: Primitive

Draws a single line.

Constructor Parameter:

attrs: An object containing values for each of the fields in the object. If the ***attrs*** parameter is not specified, or if one of the fields is not specified, use predefined defaults (defined for you).

Methods:

draw: Draw the line.

Fields (in addition to those specified by ancestors):

startX: Starting x coordinate of line. Must be >=0.

startY: Starting y coordinate of line. Must be >=0.

endX: Ending x coordinate of line. Must be >=0.

endY: Ending y coordinate of line. Must be >=0.

Rectangle

Inherits from: Primitive (implemented for you)

Draws a rectangle

Constructor Parameter:

attrs: An object containing values for each of the fields in the object. If the ***attrs*** parameter is not specified, or if one of the fields is not specified, use predefined defaults (defined for you).

Methods (in addition to those specified by ancestors):

draw: Draw the rectangle as specified by x, y, width, and height

Fields (in addition to those specified by ancestors):

x: x coordinate of the top left corner of the rectangle

y: y coordinate of the top left corner of the rectangle

width: width of the rectangle

height: height of the rectangle

Container

Inherits from: Drawable

A container is a rectangular object that can have other drawable objects as children. When drawn, the container draws itself as well as all of its children. All children inside the container are first rotated, then

translated to their (left, top) positions and drawn (note: transformations execute in reverse order, so to implement this first do translate, then rotate). The container also clips all of its content to its bounds.

Constructor Parameter:

attrs: An object containing values for each of the fields in the object. If the ***attrs*** parameter is not specified, or if one of the fields is not specified, use predefined defaults (defined for you).

Methods:

draw: Draws itself and its children. If a child is not visible (visible property set to false), does not draw it.

Fields (in addition to those specified by ancestors):

width: The width of the container
height: The height of the container
borderWidth: How wide the container's border is.
borderColor: Color of the border.
fill: Fill color of the container. Default value is "". If fill is not specified or "", do not fill in the container.
children: Drawable objects that are located within the container.

Testing Part I

We have provided code for you to test out your drawing library. The test code has two different parts. The first part (test-primitives.html) makes a drawing using only the primitives in your library (i.e. no containers). You should probably start implementing only the primitives, and make sure test-primitives.html works before moving on to the next part. The next part (test-containers.html) tests whether your container code works. test-containers.html assumes that test-primitives.html works, so you should do test this second. We have also included images with the expected output.

Part II: Make a Doodle

Make a doodle similar to a Google doodle to show off your library. Your doodle should have some text in it; however it does not need to say Google. I would recommend using your name as your doodle, for example. I will be grading your doodle based on how interesting it is. 5 points of your base grade will be based on your doodle, and you can get up to 10 points extra credit (out of a max of 10 points) for especially nice doodles.

Bells and Whistles

Completing the above requirements and having well-documented code with no errors will get you 45 out of 50 points, which is an A. You must complete one of the below 'bells and whistles' to get full points. You can complete as many of these as you want, but you will get no more than 10 points total. Different bonuses are worth different numbers of points, based on difficulty. You may receive up to 10 points for bells and whistles, bringing you up to up to 5 extra credit points. I hope these are fun for you!

1. **Implement some fancy container (5 points):** Implement a container that can be in an arbitrary shape. Make sure to include documentation that explains what you do!
2. **Enable animation (up to 10 points):** There are many ways that you could potentially add animation to your code. Depending on the complexity of the animations you enable and their extensibility, I

will assign up to 10 points for enabling animation. Make sure you thoroughly document your animation code (either in comments or a readme file).

3. **Make an awesome doodle (up to 10 points):** I will be awarding especially creative doodles extra credit.
4. **Handle Input (up to 10 points):** Make your doodle interactive by handling input (i.e. mouse or keyboard events). I will award up to 10 points for this, based on how complex and extensible your implementation is.
5. **Add parsing (10 points):** Create an XML parser that will take an XML description of a scene and either output code to draw this or draw the parser.
6. **Build a drawing application for your library (10 points):** Create an application that allows you to create drawings and then save your drawings by generating code to re-create your drawing. Your drawing should support structure (i.e. you should be able to add items in containers).

Show and Tell

We will be picking the nicest doodles and showing them to the class. They may also be put on a public website. Please let me know if you do not want your solution shown in the README file in your assignment.

Turning Your Program In

This project has the same procedure as Projects 0 and 1. Make sure to include the following files:

`doodle-library.js`	Your completed doodle library
`doodle.html`	HTML file of your doodle
`doodle.js`	JavaScript file of your doodle
`primitive-test.html`	Keep all of the test files
`primitive-test.js`	Keep all of the test files
`container-test.html`	Keep all of the test files
`container-test.js`	Keep all of the test files
`README.txt`	

The project is due Thursday, October 3, 2013 at 10:00 am. You should turn in your assignment via Blackboard, attaching a zip file with the contents below. Please write a README.txt file which mentions any online sources you used to help with the project, as well as any notes about your programs (i.e. if you couldn't get a particular part of the project to work). Name the file lastname_firstname_P#.zip. For example:

`chang_kerry_p2.zip`

Grading

Your assignment will be graded as follows:

Turn in is correct, code has no syntax errors	5 pts
Correct implementation of classes related to container	15 pts

Aesthetics of your doodle	5 pts
Use of inheritance, elegance of code.	10 pts
Correct implementation of non-container classes	15 pts
Bells and whistles	5 – 10 pts
Formatting, Comments and Coding style	5 pts