StdDraw

Robert Sedgewick and Kevin Wayne

The StdDraw class (https://introcs.cs.princeton.edu/java/stdlib/StdDraw.java.html) provides a basic capability for creating drawings with your programs. It uses a simple graphics model that allows you to create drawings consisting of points, lines, squares, circles, and other geometric shapes in a window on your computer and to save the drawings to a file. Standard drawing also includes facilities for text, color, pictures, and animation, along with user interaction via the keyboard and mouse.

This documentation, along with API documents, is online at https://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html

For additional documentation, see Section 1.5 of *Computer Science: An Interdisciplinary Approach* by Robert Sedgewick and Kevin Wayne (2017) (https://introcs.cs.princeton.edu/15inout).

More examples can br found at https://introcs.cs.princeton.edu/java/15inout/ and https://introcs.cs.princeton.edu/java/23recursion/

StdDraw.java is part of a larger set of libraries which Sedgewick and Wayne package up as stdlib.jar, available at https://introcs.cs.princeton.edu/java/code/stdlib.jar

A summary of (most of) the StdDraw API taken from *Computer Science: An Interdisciplinary Approach*:

public class StdDraw	
drawing commands	
void line(double x0, double y0, double x1, double y1)	
void point(double x, double y)	
void circle(double x, double y, double radius)	
void filledCircle(double x, double y, double radius)	
void square(double x, double y, double radius)	
void filledSquare(double x, double y, double radius)	
void rectangle(double x, double y, double r1, double r2)	
void filledRectangle(double x, double y, double r1, double r2)	
void polygon(double[] x, double[] y)	
void filledPolygon(double[] x, double[] y)	
void text(double x, double y, String s)	
control commands	
void setXscale(double x0, double x1)	*reset x-scale to (x0, x1)*
void setYscale(double y0, double y1)	*reset y-scale to (y0, y1)*
void setPenRadius(double radius)	*set pen radius to* `radius`
void setPenColor(Color color)	*set pen color to* `color`
void setFont(Font font)	*set text font to* `font`
void setCanvasSize(int w, int h)	*set canvas size to w-by-h*
void enableDoubleBuffering()	*enable double buffering*
void disableDoubleBuffering()	*disable double buffering*
void show()	*copy the offscreen canvas to the onscreen canvas*
void clear(Color color)	*clear the canvas to color* `color`
void pause(int dt)	*pause* `dt` *milliseconds*
void save(String filename)	*save to a* `.jpg` *or* `.png` *file*

Note: Methods with the same names but no arguments reset to default values.

API for our library of static methods for standard drawing

The rest of this document describes the API in full detail.

Points and lines

You can draw points and line segments with the following methods:

- `point(double x, double y)`
- `line(double x1, double y1, double x2, double y2)`

The x- and y-coordinates must be in the drawing area (between 0 and 1 and by default) or the points and lines will not be visible.

Squares, circles, rectangles, and ellipses

You can draw squares, circles, rectangles, and ellipses using the following methods:

- `circle(double x, double y, double radius)`
- `ellipse(double x, double y, double semiMajorAxis, double semiMinorAxis)`
- `square(double x, double y, double halfLength)`
- `rectangle(double x, double y, double halfWidth, double halfHeight)`

All of these methods take as arguments the location and size of the shape. The location is always specified by the x- and y-coordinates of its center.

The size of a circle is specified by its radius and the size of an ellipse is specified by the lengths of its semi-major and semi-minor axes.

The size of a square or rectangle is specified by its half-width or half-height. The convention for drawing squares and rectangles is parallel to those for drawing circles and ellipses, but may be unexpected to the uninitiated.

The methods above trace outlines of the given shapes. The following methods draw filled versions:

- `filledCircle(double x, double y, double radius)`
- `filledEllipse(double x, double y, double semiMajorAxis, double semiMinorAxis)`
- `filledSquare(double x, double y, double radius)`
- `filledRectangle(double x, double y, double halfWidth, double halfHeight)`

Circular arcs

You can draw circular arcs with the following method:

- `arc(double x, double y, double radius, double angle1, double angle2)`

The arc is from the circle centered at (x, y) of the specified radius.

The arc extends from angle1 to angle2. By convention, the angles are polar (counterclockwise angle from the x-axis) and represented in degrees. For example, `StdDraw.arc(0.0, 0.0, 1.0, 0, 90)` draws the arc of the unit circle from 3 o'clock (0 degrees) to 12 o'clock (90 degrees).

Polygons

You can draw polygons with the following methods:

- `polygon(double[] x, double[] y)`
- `filledPolygon(double[] x, double[] y)`

The points in the polygon are (x[i], y[i]).

For example, the following code fragment draws a filled diamond with vertices (0.1, 0.2), (0.2, 0.3), (0.3, 0.2), and (0.2, 0.1):

```
double[] x = { 0.1, 0.2, 0.3, 0.2 };
double[] y = { 0.2, 0.3, 0.2, 0.1 };
StdDraw.filledPolygon(x, y);
```

Pen size

The pen is circular, so that when you set the pen radius to r and draw a point, you get a circle of radius r. Also, lines are of thickness 2r and have rounded ends. The default pen radius is 0.005 and is not affected by coordinate scaling. This default pen radius is about 1/200 the width of the default canvas, so that if you draw 100 points equally spaced along a horizontal or vertical line, you will be able to see individual circles, but if you draw 200 such points, the result will look like a line.

- `setPenRadius(double radius)`

For example, `StdDraw.setPenRadius(0.025)` makes the thickness of the lines and the size of the points to be five times the 0.005 default.

To draw points with the minimum possible radius (one pixel on typical displays), set the pen radius to 0.0.

Pen color

All geometric shapes (such as points, lines, and circles) are drawn using the current pen color. By default, it is black. You can change the pen color with the following methods:

- `setPenColor(int red, int green, int blue)`
- `setPenColor(Color color)`

The first method allows you to specify colors using the RGB color system.

This http://johndyer.name/lab/colorpicker/ color picker is a convenient way to find a desired color.

The second method allows you to specify colors using the Color data type, or you can use this method with one of these predefined colors in standard drawing: BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW, BOOK_BLUE, BOOK_LIGHT_BLUE, BOOK_RED, and PRINCETON_ORANGE.

For example, `StdDraw.setPenColor(StdDraw.MAGENTA)` sets the pen color to magenta.

Window title

By default, the stanard drawing window title is "Standard Draw". You can change the title with the following method:

- `setTitle(String title)`

This sets the standard drawing window title to the specified string.

Canvas size

By default, all drawing takes places in a 512-by-512 canvas. The canvas does not include the window title or window border. You can change the size of the canvas with the following method:

- `setCanvasSize(int width, int height)`

This sets the canvas size to be width-by-height pixels. It also erases the current drawing and resets the coordinate system, pen radius, pen color, and font back to their default values.

Ordinarly, this method is called once, at the very beginning of a program. For example, `StdDraw.setCanvasSize(800, 800)` sets the canvas size to be 800-by-800 pixels.

Canvas scale and coordinate system

By default, all drawing takes places in the unit square, with (0, 0) at lower left and (1, 1) at upper right. You can change the default coordinate system with the following methods:

- `setXscale(double xmin, double xmax)`
- `setYscale(double ymin, double ymax)`
- `setScale(double min, double max)`

The arguments are the coordinates of the minimum and maximum x- or y-coordinates that will appear in the canvas. For example, if you wish to use the default coordinate system but leave a small margin, you can call `StdDraw.setScale(-.05, 1.05)`.

These methods change the coordinate system for subsequent drawing commands; they do not affect previous drawings.These methods do not change the canvas size; so, if the x- and y-scales are different, squares will become rectangles and circles will become ellipses.

Text

You can use the following methods to annotate your drawings with text:

- `text(double x, double y, String text)`
- `text(double x, double y, String text, double degrees)`
- `textLeft(double x, double y, String text)`
- `textRight(double x, double y, String text)`

The first two methods write the specified text in the current font, centered at (x, y).

The second method allows you to rotate the text.

The last two methods either left- or right-align the text at (x, y).

The default font is a Sans Serif font with point size 16. You can use the following method to change the font:

- `setFont(Font font)`

You use the Font data type to specify the font. This allows you to choose the face, size, and style of the font. For example, the following code fragment sets the font to Arial Bold, 60 point.

```
Font font = new Font("Arial", Font.BOLD, 60);
StdDraw.setFont(font);
```

```
StdDraw.text(0.5, 0.5, "Hello, World");
```

Images

You can use the following methods to add images to your drawings:

- `picture(double x, double y, String filename)`
- `picture(double x, double y, String filename, double degrees)`
- `picture(double x, double y, String filename, double scaledWidth, double scaledHeight)`
- `picture(double x, double y, String filename, double scaledWidth, double scaledHeight, double degrees)`

These methods draw the specified image, centered at (x, y). The supported image formats are JPEG, PNG, and GIF.

The image will display at its native size, independent of the coordinate system. Optionally, you can rotate the image a specified number of degrees counterclockwise or rescale it to fit snugly inside a width-by-height bounding box.

Saving to a file

You save your image to a file using the `File → Save` menu option. You can also save a file programatically using the following method:

- `save(String filename)`

The supported image formats are JPEG and PNG. The filename must have either the extension .jpg or .png.

We recommend using PNG for drawing that consist solely of geometric shapes and JPEG for drawings that contains pictures.

Clearing the canvas

To clear the entire drawing canvas, you can use the following methods:

- `clear()`
- `clear(Color color)`

The first method clears the canvas to white; the second method allows you to specify a color of your choice. For example, `StdDraw.clear(StdDraw.LIGHT_GRAY)` clears the canvas to a shade of gray.

Computer animations and double buffering

Double buffering is one of the most powerful features of standard drawing, enabling computer animations.

The following methods control the way in which objects are drawn:

- `enableDoubleBuffering()`
- `disableDoubleBuffering()`

- `show()`
- `pause(int t)`

By default, double buffering is disabled, which means that as soon as you call a drawing method – such as point() or line() – the results appear on the screen.

When double buffering is enabled by calling `enableDoubleBuffering()`, all drawing takes place on the offScr canvas. The offScr canvas is not displayed. Only when you call `show()` does your drawing get copied from the offScr canvas to the onscreen canvas, where it is displayed in the standard drawing window. You can think of double buffering as collecting all of the lines, points, shapes, and text that you tell it to draw, and then drawing them all simultaneously, upon request.

The most important use of double buffering is to produce computer animations, creating the illusion of motion by rapidly displaying static drawings. To produce an animation, repeat the following four steps:

1. Clear the offScr canvas.
2. Draw objects on the offScr canvas.
3. Copy the offScr canvas to the onscreen canvas.
4. Wait for a short while.

The `clear()`, `show()`, and `pause(int t)` methods support the first, third, and fourth of these steps, respectively. For example, this code fragment animates two balls moving in a circle.

```
StdDraw.setScale(-2, +2);
StdDraw.enableDoubleBuffering();
for (double t = 0.0; true; t += 0.02) {
 double x = Math.sin(t);
 double y = Math.cos(t);
 StdDraw.clear();
 StdDraw.filledCircle(x, y, 0.05);
 StdDraw.filledCircle(-x, -y, 0.05);
 StdDraw.show();
 StdDraw.pause(20);
}
```

Keyboard and mouse inputs

Standard drawing has very basic support for keyboard and mouse input. It is much less powerful than most user interface libraries provide, but also much simpler.

You can use the following methods to intercept mouse events:

- `isMousePressed()`
- `mouseX()`
- `mouseY()`

The first method tells you whether a mouse button is currently being pressed. The last two methods tells you the x- and y-coordinates of the mouse's current position, using the same coordinate system as the canvas (the unit square, by default).

You should use these methods in an animation loop that waits a short while before trying to poll the mouse for its current state.

You can use the following methods to intercept keyboard events:

- `hasNextKeyTyped()`
- `nextKeyTyped()`
- `isKeyPressed(int keycode)`

If the user types lots of keys, they will be saved in a list until you process them.

The first method tells you whether the user has typed a key that your program has not yet processed).

The second method returns the next key that the user typed (that your program has not yet processed) and removes it from the list of saved keystrokes.

The third method tells you whether a key is currently being pressed.

Accessing control parameters

You can use the following methods to access the current pen color, pen radius, and font:

- `getPenColor()`
- `getPenRadius()`
- `getFont()`

These methods are useful when you want to temporarily change a control parameter and reset it back to its original value.

Corner cases

Drawing an object outside (or partly outside) the canvas is permitted. However, only the part of the object that appears inside the canvas will be visible.

Any method that is passed a null argument will throw an `IllegalArgumentException`.

Any method that is passed a `Double.NaN`, `Double.POSITIVE_INFINITY`, or `Double.NEGATIVE_INFINITY` argument will throw an `IllegalArgumentException`.

Due to floating-point issues, an object drawn with an x- or y-coordinate that is way outside the canvas (such as the line segment from (0.5, -10^{308}) to (0.5, 10^{308}) may not be visible even in the part of the canvas where it should be.

Performance tricks

Use *double buffering* for static drawing with a large number of objects. That is, call `enableDoubleBuffering()` before the sequence of drawing commands and call `show()` afterwards.

Incrementally displaying a complex drawing while it is being created can be intolerably inefficient on many computer systems.

When drawing computer animations, call `show()` only once per frame, not after drawing each individual object.

If you call `picture()` multiple times with the same filename, Java will cache the image, so you do not incur the cost of reading from a file each time.

Known bugs and issues

The `picture()` methods may not draw the portion of the image that is inside the canvas if the center point (x, y) is outside the canvas. This bug appears only on some systems.

Todo

- Add support for gradient fill, etc.
- Fix `setCanvasSize()` so that it can only be called once.
- On some systems, drawing a line (or other shape) that extends way beyond canvas (e.g., to infinity) dimensions does not get drawn.

Remarks

- don't use `AffineTransform` for rescaling since it inverts images and strings